お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

User's Manual

RENESAS

M3T-MR308/4 V.4.00

ユーザーズマニュアル

M16C/70,80,M32C/80 シリーズ用リアルタイムOS

誤記に関するお詫び:
本資料 P.116 【割り込みベクタ定義】の記載事項に誤記があり、訂正いたしました。

ルネサス エレクトロニクス
www.renesas.com

Rev.2.00 2005.11

- Microsoft、MS-DOS、WindowsおよびWindows NTは、米国Microsoft Corporationの米国およびその他の国における登録商標です。
- HP-UXは、米国Hewlett-Packard Companyのオペレーティングシステムの名称です。
- Sun 、Java およびすべてのJava関連の商標およびロゴは、米国およびその他の国における米国Sun Microsystems, Inc.の商標または登録商標です。
- UNIXは、X/Open Company Limitedが独占的にライセンスしている米国ならびに他の国における登録商標です。
- IBMおよびATは、米国International Business Machines Corporationの登録商標です。
- HP 9000は、米国Hewlett-Packard Companyの商品名称です。
- SPARCおよびSPARCstationは、米国SPARC International, Inc.の登録商標です。
- Intel, Pentiumは、米国Intel Corporationの登録商標です。
- AdobeおよびAcrobatは、Adobe Systems Incorporated（アドビシステムズ社）の登録商標です。
- NetscapeおよびNetscape Navigatorは、米国およびその他の諸国のNetscape Communications Corporation社の登録商標です。
- TRONは、"The Real-Time Operating System Nucleus" の略称です。
- ITRONは、"Industrial TRON" の略称です。
- μITRONは、"Micro Industrial TRON" の略称です。μITRON仕様の著作権は(社)トロン協会に属しています。
- TRON、ITRON、およびμITRONは、コンピュータの仕様に対する名称であり、特定の商品ないし商品群を指すものではありません。
- その他すべてのブランド名および製品名は個々の所有者の登録商標もしくは商標です。

安全設計に関するお願い

- 弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

本資料ご利用に際しての留意事項

- 本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報について株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズが所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
- 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズは責任を負いません。
- 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズは、予告なしに、本資料に記載した製品又は仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりましては、事前に株式会社ルネサス テクノロジ、株式会社ルネサス ソリューションズ、株式会社ルネサス販売又は特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ（http://www.renesas.com）などを通じて公開される情報に常にご注意ください。
- 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズはその責任を負いません。
- 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズは、適用可否に対する責任は負いません。
- 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、株式会社ルネサス テクノロジ、株式会社ルネサス ソリューションズ、株式会社ルネサス販売又は特約店へご照会ください。
- 本資料の転載、複製については、文書による株式会社ルネサス テクノロジおよび株式会社ルネサス ソリューションズの事前の承諾が必要です。
- 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたら株式会社ルネサス テクノロジ、株式会社ルネサス ソリューションズ、株式会社ルネサス販売又は特約店までご照会ください。

製品の内容及び本書についてのお問い合わせ先

インストーラが生成する以下のテキストファイルに必要事項を記入の上、ツール技術サポート窓口 support_tool@renesas.com まで送信ください。

¥SUPPORT¥製品名¥SUPPORT.TXT

株式会社ルネサス ソリューションズ
ツール技術サポート窓口　support_tool@renesas.com
ユーザ登録窓口　regist_tool@renesas.com
ホームページ　http://www.renesas.com/jp/tools

はじめに

M3T-MR308/4(以下 MR308 と略す)は M16C/70,80,M32C/80 シリーズ用のリアルタイム・オペレーティングシステム[1]です。MR308 はμITRON4.0 仕様[2]に準拠しています。

本マニュアルは MR308 を使用したプログラムの作成手順および作成上の注意事項について説明します。各サービスコールの詳細な使用方法については「MR308 リファレンスマニュアル」を参照してください。

MR308 を使うために必要なこと

MR308 を使用したプログラムを作成するには弊社下記製品を別途御購入して頂く必要があります。

- M16C/70,80,M32C/80 シリーズ用 C コンパイラパッケージ M3T-NC308WA(以下 NC308WA と略す)

ドキュメント一覧

MR308 に添付されているドキュメントは以下の 3 種類あります。

- リリースノート
  ソフトウェアの概要やユーザーズマニュアル、リファレンスマニュアルの訂正などを記載したドキュメントです。
- ユーザーズマニュアル (PDF ファイル)
  MR308 を使用したプログラムの作成手順や作成上の注意事項を記載したドキュメントです。
- リファレンスマニュアル (PDF ファイル)
  MR308 のサービスコールの使用方法や使用例を記述したドキュメントです。
  本マニュアルを読む前に必ずリリースノートをお読みください。

ソフトウェアの使用権

ソフトウェアの使用権はソフトウェア使用権許諾契約書に基づきます。MR308 はお客様の製品開発の目的でのみ使用できます。その他の目的での使用はできませんのでご注意ください。

また、本マニュアルによってソフトウェアの使用権の実施に対する保証及び使用権の実施の許諾を行うものではありません。

---

[1] 以降リアルタイム OS と略します。

[2] ・μITRON4.0 仕様は、(社)トロン協会が策定したオープンなリアルタイムカーネル仕様です。
・μITRON4.0 仕様の仕様書は、(社)トロン協会ホームページ（http://www.assoc.tron.org/）から入手が可能です。
・μITRON 仕様の著作権は(社)トロン協会に属しています。

目次

図目次

表目次

第 1 章　　ユーザーズマニュアルの構成

MR308 ユーザーズマニュアルは、９つの章から構成されています。

- 第 2 章 概要
  MR308 の目的や、概略の機能、位置づけなどを説明します。
- 第 3 章 MR308 入門
  MR308 を使用する上で必要となる考え方や用語などを説明します。
- 第 4 章 アプリケーション作成手順概要
  MR308 を使用してアプリケーションプログラムを作成する場合の開発手順の概要を説明します。
- 第 5 章 アプリケーション作成手順詳細
  MR308 を使用してアプリケーションプログラムを作成する場合の開発手順を詳細に説明します。
- 第 6 章 コンフィギュレータの使用方法
  コンフィギュレーションファイルの記述方法、および、コンフィギュレータの使用方法を詳細に説明します。
- 第 7 章 アプリケーション作成の手引き
  MR308 を使用してアプリケーションプログラムを作成する際に知っておいたほうがよい事項や、注意事項について説明します。
- 第 8 章 サンプルプログラムの説明
  製品にソースファイル形式で含まれている MR308 サンプルアプリケーションプログラムについて説明します。
- 第 9 章 別 ROM 化について
  別 ROM 化の方法について説明します。

第 2 章　　概要

2.1 MR308 のねらい

近年マイクロコンピュータの急激な進歩にともない､マイクロコンピュータ応用製品の機能が複雑化してきています。これにともない、マイクロコンピュータのプログラムサイズが大きくなってきています。また製品開発競争が激化しマイクロコンピュータ応用製品を短期間に開発しなければなりません。すなわち、マイクロコンピュータのソフトウェアを開発している技術者は今までより大きなプログラムを今までより短期間で開発することが要求されてきます。そこでこの困難な要求を解決するためには以下のことを考えていかなければなりません。

1. **ソフトウェアの再利用性を高めて、 開発すべきソフトウェアの量を削減する。**

   このためにはソフトウェアをできるだけ機能単位で独立したモジュールに分割して再利用できるようにする方法があります。すなわち、汎用サブルーチン集などを多く蓄積してそれをプログラム開発時に使用します。ただこの方法では、時間やタイミングに依存したプログラムは再利用するのは困難です。ところが実際の応用プログラムは時間やタイミングに依存したプログラムがかなりの部分を占めていてこのような手法で再利用できるプログラムはあまり多くありません。

2. **チームプログラミングを推進し、1 つのソフトウェアを何名かの技術者でおこなうようにする。**

   チームプログラミングをおこなうには色々な問題があります。1 つはデバッグ作業をおこなうにあたり、チームプログラミングをおこなっている技術者全員のソフトウェアがデバッグできる状態にないとデバッグに入れません。また、チーム内の意志統一を十分におこなう必要があります。

3. **ソフトウェアの生産効率を向上させ、技術者 1 名あたりの開発可能量を増加させる。**

   1 つは技術者の教育をおこない技術者のスキルアップをはかる方法があります。また、構造化記述アセンブラや C コンパイラなどを用いることにより、より簡単にプログラムを作成できるようにする方法があります。また、ソフトウェアのモジュール化を推進してデバッグの効率を向上させる方法等があります。

しかし、このような問題を解決するには従来の手法では限界があります。そこでリアルタイム OS[3]という新しい手法の導入が必要になってきます。

そこで、弊社はこの要求に答えるべく 16/32 ビットマイクロプロセッサ M16C/70,M16C/80,M32C/80 シリーズ用にリアルタイム OS MR308 を開発しました。MR308 を導入することにより以下のような効果があります。

4. **ソフトウェアの再利用が容易になります。**

   リアルタイム OS を導入することにより、リアルタイム OS を介してタイミングをとり、タイミングに依存したプログラムが再利用できるようになります。
   また、プログラムをタスクというモジュールに分割しますので、自然と構造化プログラミングをおこなうようになります。すなわち再利用可能なプログラムを自然に作成するようになります。

5. **チームプログラミングがおこないやすくなります。**

   リアルタイム OS を導入することにより、プログラムがタスクという機能単位のモジュールに分割されますので、タスク単位で開発をおこなう技術者を振り分け開発からタスク単位でデバッグまでできるようになります。とくにリアルタイム OS を導入すると、プログラムが全てでき上がっていなくてもタスクさえ出来ていればその部分のデバッグを初めることが容易にできます。またタスク単位で技術者を割り振ることができますので、作業分担が容易におこなえます。

6. **ソフトウェアの独立性が高くなり、 プログラムをデバックしやすくなります。**

   リアルタイム OS を導入することにより、プログラムをタスクという独立した小さなモジュールに分割できますので、プログラムをデバッグする際ほとんどはその小さなモジュールに着目するだけでデバッグすることができます。

7. **タイマ制御が簡単になります。**

   従来例えば、10ms ごとにある処理を動作させるためには、マイクロコンピュータのタイマ機能を用いて定期的に割り込みを発生させて処理させていました。ところが、マイクロコンピュータのタイマの数には限りがありますのでタイマが足らなくなったら 1 本のタイマを複数の処理に使用するなどの手法を用いて解決していました。

[3] OS：Operating System

ところがリアルタイム OS を導入することにより、リアルタイム OS の時間管理機能を使用して一定時間毎にある処理をさせるというプログラムを、マイクロコンピュータのタイマ機能を特に意識せずに作成することができます。また、同時にプログラマから見たとき疑似的にマイクロコンピュータに無限本数のタイマが搭載されたようにプログラムを作成することができます。

8. ソフトウェアの保守性が向上します。

リアルタイム OS を導入することにより開発したソフトウェアが小さなタスクと呼ばれるプログラムの集合で構成されます。これにより開発完了後保守をおこなう場合、小さなタスクだけを保守すればよくなり保守性が向上します。

9. ソフトウェアの信頼性が向上します。

リアルタイム OS を導入することにより、プログラムの評価、試験などがタスクという小さなモジュール単位でおこなえますので評価、試験が容易になりひいては信頼性が向上します。

10. マイクロコンピュータの性能を最大限生かすことができます。これにより応用製品の性能向上が望めます。

リアルタイム OS を導入することにより、入出力待ちなどのマイクロコンピュータのむだな動作を減少させることができます。これによりマイクロコンピュータの能力を最大限に引き出すことができます。ひいては応用製品の性能向上につながります。

2.2 TRON 仕様と MR308

TRON 仕様とは The Real-Time Operating system Nucleus 仕様の略で、リアルタイム・オペレーティングシステムの核となる部分の仕様を意味します。TRON 仕様の設計を中心とした TRON プロジェクトは、東京大学の坂村健博士を中心として進められています。

この TRON プロジェクトで推進されているものの 1 つに ITRON 仕様があります。ITRON 仕様は Industrial TRON 仕様の略で、産業用組み込みシステムをターゲットとしたリアルタイム・オペレーティングシステムの仕様です。

ITRON 仕様は広い分野の応用に十分対応できるように数多くの機能を有しています。このため ITRON システムは比較的大きなメモリ容量と処理能力を必要とします。μITRON 2.0 は、1989 年に仕様が規定されたもので、処理速度を向上させるため仕様を整理し、必要十分な機能のみにサブセット化されたものです。

1993 年には、μITRON 2.0 と ITRON 仕様を統合し、接続機能を追加したμITRON 3.0 が規定されました。

さらに、1999 年には、互換性の強化を図ったμITRON 4.0[4]が規定されました。

MR308 は、μITRON 4.0 仕様に準拠した 16/32 ビットマイクロコンピュータ M16C/70,80,M32C/80 シリーズ用に開発された、リアルタイム・オペレーティングシステムです。μITRON 4.0 仕様では、ソフトウェアの移植性を確保するための試みとしてスタンダードプロファイルを規定していますが、MR308 は、スタンダードプロファイルのうち、静的 API およびタスク例外を除くすべてのサービスコールをインプリメントしています。

[4] ・μITRON4.0 仕様は、(社)トロン協会が策定したオープンなリアルタイムカーネル仕様です。
・μITRON4.0 仕様の仕様書は、(社)トロン協会ホームページ（http://www.assoc.tron.org/）から入手が可能です。

2.3 MR308 の特長

MR308 は以下に示す特長を持っています。

1. μITRON 仕様に準拠したリアルタイム・オペレーティングシステム

   MR308 は ITRON 仕様をワンチップマイクロコンピュータでも実装できるように機能を整理したμITRON 仕様に基づいて開発されました。

   μITRON 仕様は ITRON 仕様のサブセットですので ITRON 教科書として出版されている文献や ITRON セミナー等で得た知識をほとんどそのまま役立てることができます。また、ITRON 仕様に準拠したリアルタイム OS を用いて開発したアプリケーションプログラムを MR308 に移行するのは比較的容易に行えます。

2. 高速処理を実現

   マイコンのアーキテクチャを活用し、高速処理を実現しています。

3. 必要モジュールのみを自動選択することにより常に最小サイズのシステムを構築

   MR308 は M16C/70,80,M32C/80 シリーズオブジェクトライブラリ形式で供給されています。

   したがって、リンケージエディタ LN308 のもつ機能により、数ある MR308 の機能モジュールのなかで使用しているモジュールのみを自動選択してシステムを生成します。このため、常に最小サイズのシステムが自動的に生成されます。

4. C コンパイラ NC308WA を用いて C 言語でアプリケーションプログラムが開発可能

   C コンパイラ NC308WA を用いて、MR308 のアプリケーションプログラムを C 言語で開発できます。また C 言語から MR308 の機能を呼び出すためのインタフェースライブラリが製品に添付されています。

5. 上流工程ツール "コンフィギュレータ" により、容易な開発手順

   ROM 書き込み形式ファイルまでの作成を簡単な定義のみでおこなえるコンフィギュレータを装備しています。これにより、どんなライブラリを結合する必要があるかなどを特に気にする必要はありません。

   また、M3T-MR308/4 V.4.0 Release 00 より、GUI 化されたコンフィギュレータを用意しました。これにより、コンフィギュレーションファイルの記述形式を習得しなくとも、コンフィギュレーションが可能になりました。

第 3 章　　MR308 入門

3.1 リアルタイム OS の考え方

本節では、リアルタイム OS の基本概念について説明します。

3.1.1 リアルタイム OS の必要性

近年半導体技術の進歩にともなってシングルチップマイクロコンピュータ（マイコン)の ROM 容量が増大してきています。

このような大 ROM 容量のマイクロコンピュータの出現によりそのプログラム開発が従来の方法では困難になってきています。図 3.1にプログラムサイズと開発期間（開発の困難さ）との関係を示します。この図 3.1はあくまでイメージ図ですが、プログラムのサイズが大きくなるに従い開発期間が指数関数的に長くなってきます。

例えば 32K バイトのプログラムを 1 個開発するより、8K バイトのプログラムを 4 個開発する方が簡単です。[5]

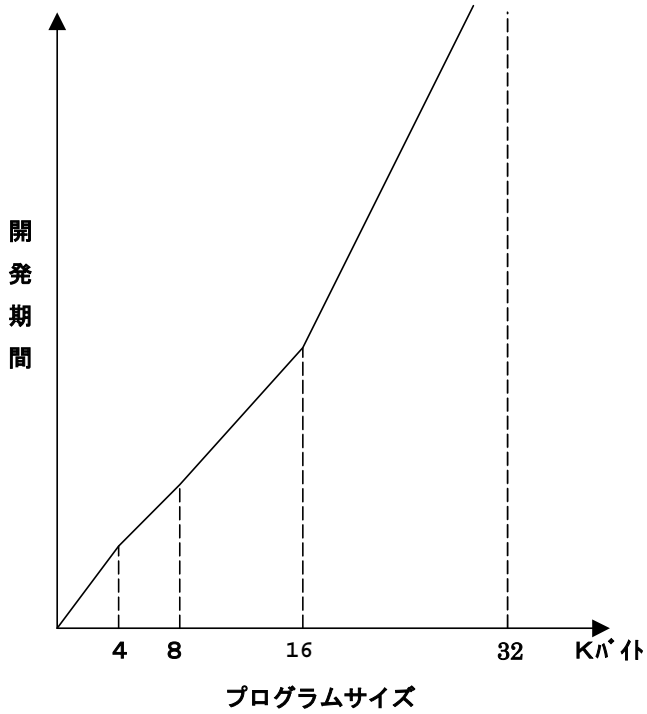


図 3.1　プログラムサイズと開発期間

そこで大きなプログラムを短期間に簡単に開発するための手法が必要になってきます。この方法として小さな ROM 容量のマイクロコンピュータを多く使う方法があります。たとえば、図 3.2にオーディオ機器システムを複数のマイクロコンピュータで構成した例を示します。

[5] ROM 詰めの必要がないことを前提とします

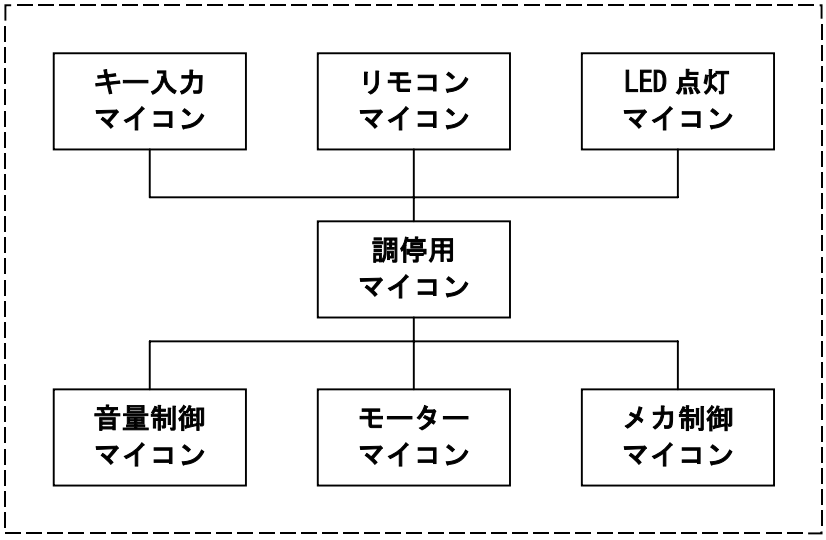


図 3.2　マイコンを多く使ったシステム例（オーディオ機器）

このように機能単位で別々のマイクロコンピュータを用いることは以下の利点があります。

1. ひとつひとつのプログラムが小さくなり、プログラム開発が容易になる。
2. 一度開発したソフトウェアを再利用することが非常に容易になる。[6]
3. 完全に機能ごとにプログラムが分離するので複数の技術者でプログラム開発が容易にできる。

この反面以下のような欠点があります。

1. 部品点数が多くなり製品の原価を上昇させる。
2. ハードウェア設計が複雑になる。
3. 製品の物理的サイズが大きくなる。

そこでそれぞれのマイクロコンピュータで動作しているプログラムを、1 つのマイクロコンピュータでソフトウェア的に、別々のマイクロコンピュータで動作しているように見せることのできるリアルタイム OS を採用すれば、上記の利点を残したままで欠点をすべて無くすことができます。

図 3.3に、図 3.2に示したシステムにリアルタイム OS を導入した場合のシステム例を示します。

[6] 例えば、図 3.2において、リモコンマイコンを別の製品にそのまま使用することができる。

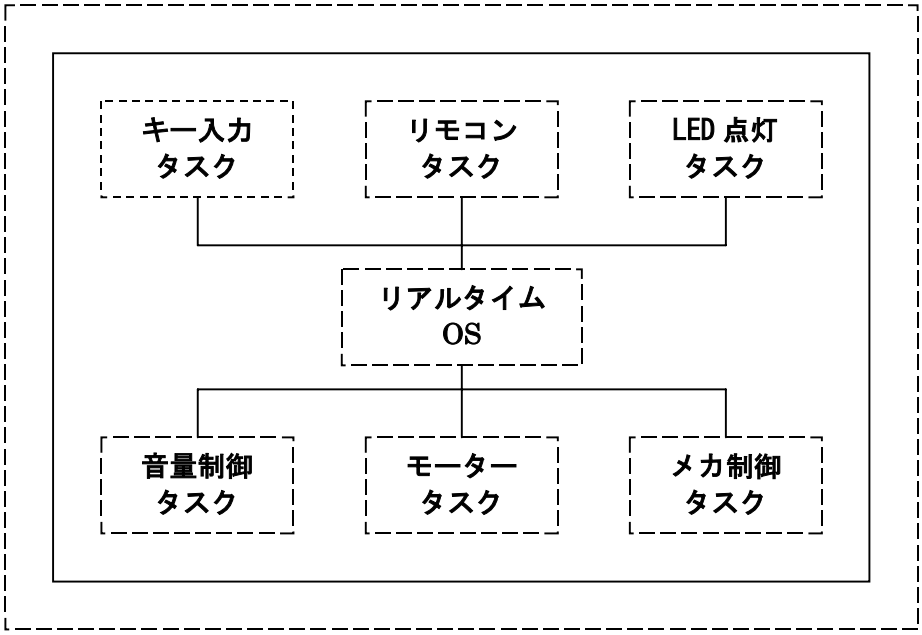


図 3.3　リアルタイム OS の導入システム例（オーディオ機器）

すなわちリアルタイム OS とは 1 個のマイクロコンピュータを、あたかも複数のマイクロコンピュータが動作しているように見せるソフトウェアです。複数のマイクロコンピュータに相当するひとつひとつのプログラムをリアルタイム OS 用語でタスクと呼びます。

3.1.2 リアルタイム OS の動作原理

リアルタイム OS は 1 個のマイクロコンピュータを、あたかも複数のマイクロコンピュータが動作しているように見せることのできるソフトウェアです。では 1 個のマイクロコンピュータをどのようにして複数あるように見せかけるのでしょうか？

それは、図 3.4に示すようにそれぞれのタスクを時分割で動作させるからです。つまり実行するタスクを一定時間ごとに切り替えて、複数のタスクが同時に実行しているように見せるのです。

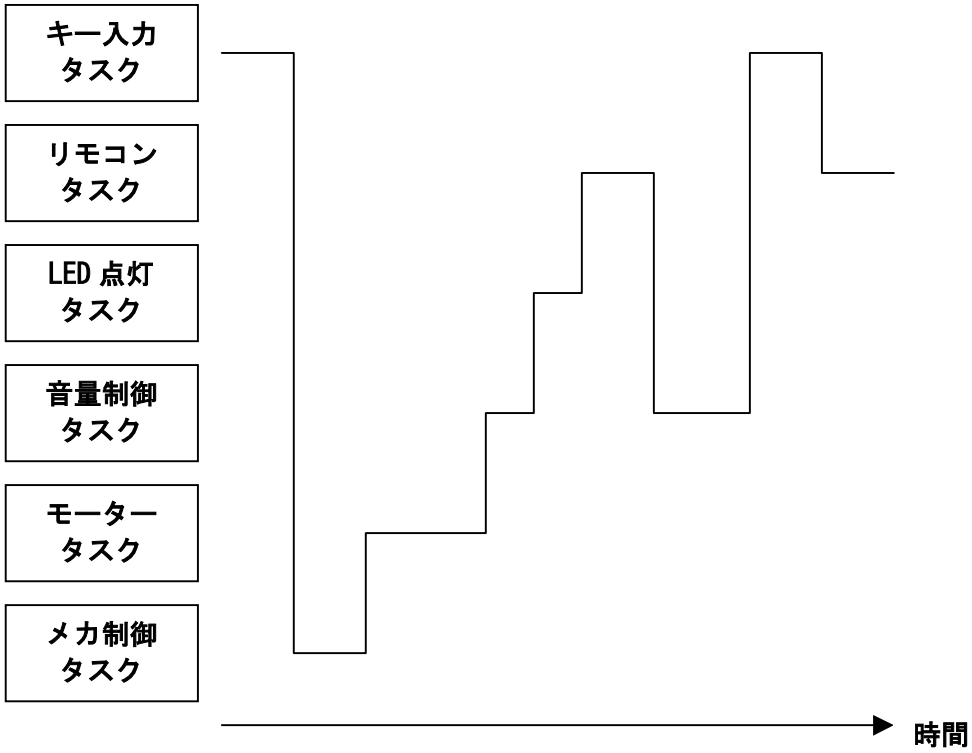


図 3.4　タスクの時分割動作

このようにタスクを一定時間ごとに切り替えて実行しています。このタスクを切り替えることをリアルタイム OS 用語でディスパッチと呼ぶこともあります。タスク切り替え (ディスパッチ)が発生する要因として以下のものがあります。

- 自分自身で切り替えを要求する。
- 割り込みなどの外的要因で切り替わる。

タスク切り替えが発生し、再度、そのタスクを実行するときには、中断していたところから再開します。 (図 3.5参照)

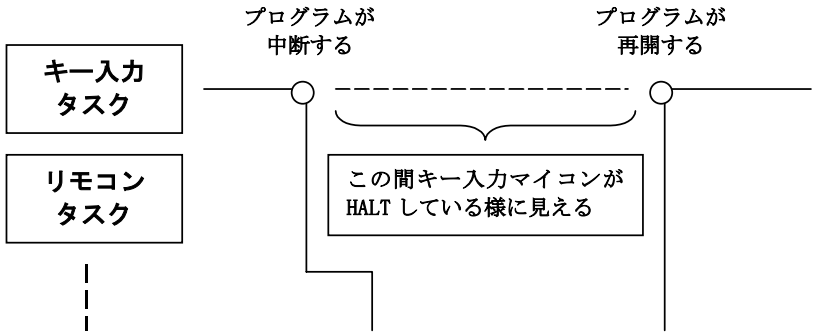


図 3.5　タスクの中断と再開

図 3.5においてキー入力タスクは、 他のタスクに実行制御が移っている間、プログラマから見ればプログラムが中断しそのマイコンが HALT しているようにみえます。

タスクの実行は、中断した時点のレジスタ内容を復帰することにより、中断した時点の状態で再開されます。すなわちタスクの切り替えとは、現在実行中のタスクのレジスタの内容をそのタスクを管理するメモリ領域に退避し、切り替えるタスクのレジスタ内容を復帰することです。

すなわちリアルタイム OS を実現するには、タスクごとにレジスタを管理し、切り換えが発生する度にそのレジスタ内容を入れ換えることにより複数のマイクロコンピュータが存在しているように見せてやればよいということになります。（図 3.6参照）。

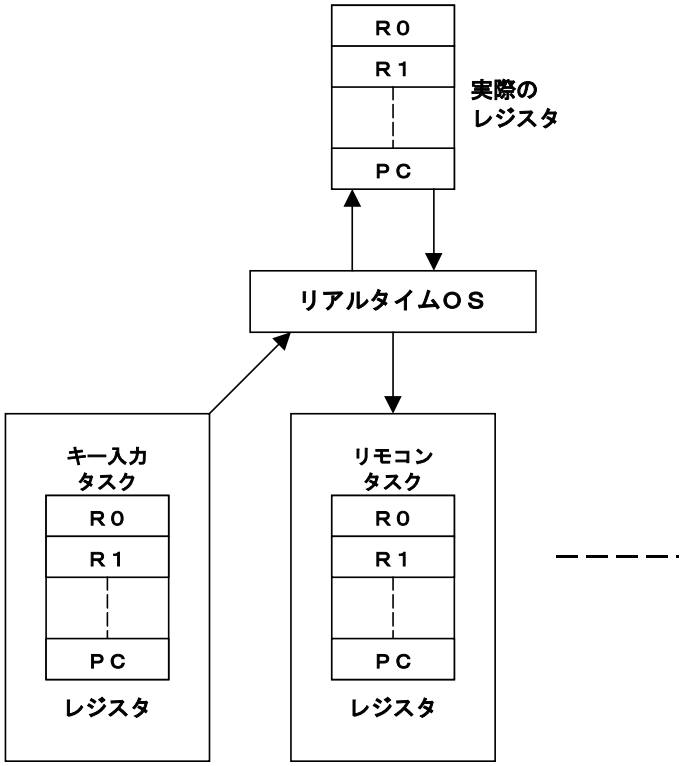


図 3.6　タスクの切り替え

図 3.7[7]は各タスクのレジスタをどのように管理しているか具体的に示したものです。

実際にはタスクごとに持つ必要のあるのはレジスタだけでなく、スタック領域もタスクごとに持つ必要があります。

[7] タスクのスタック領域をすべて同じセクションに配置した場合の図です。

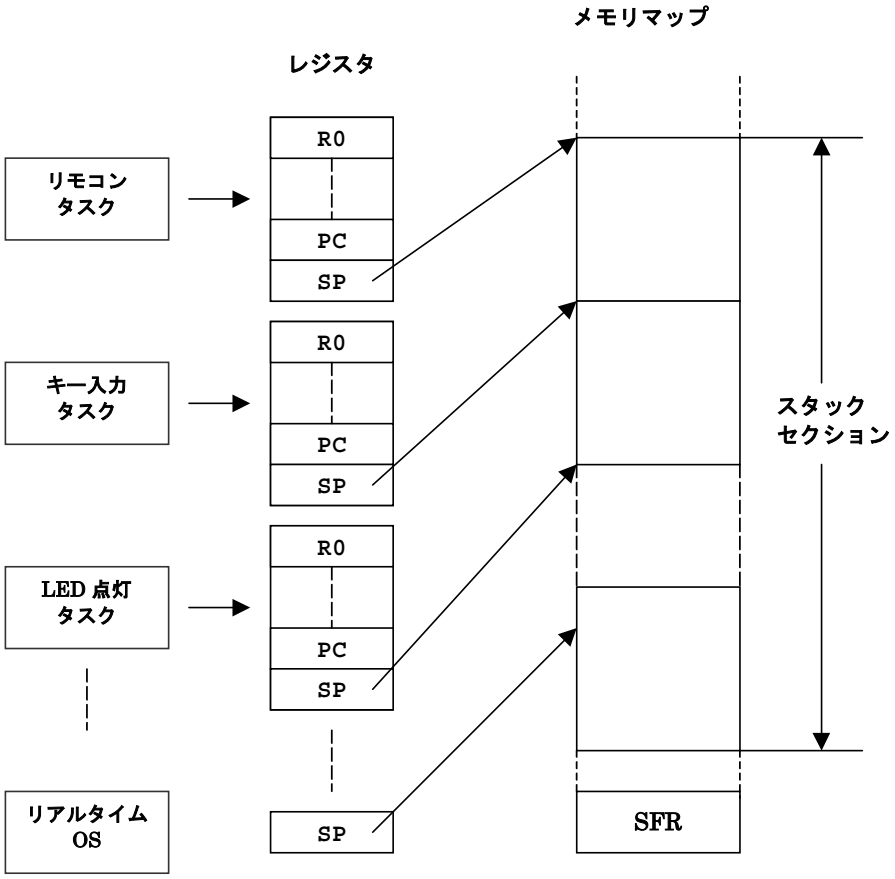


図 3.7　タスクのレジスタ領域

図 3.8は各タスクのレジスタおよびスタック領域を詳細に説明したものです。MR308 では各タスクのレジスタは図 3.8に示すようにスタック領域の中に格納され管理されています。図 3.8は、レジスタ格納後の状態を示しています。

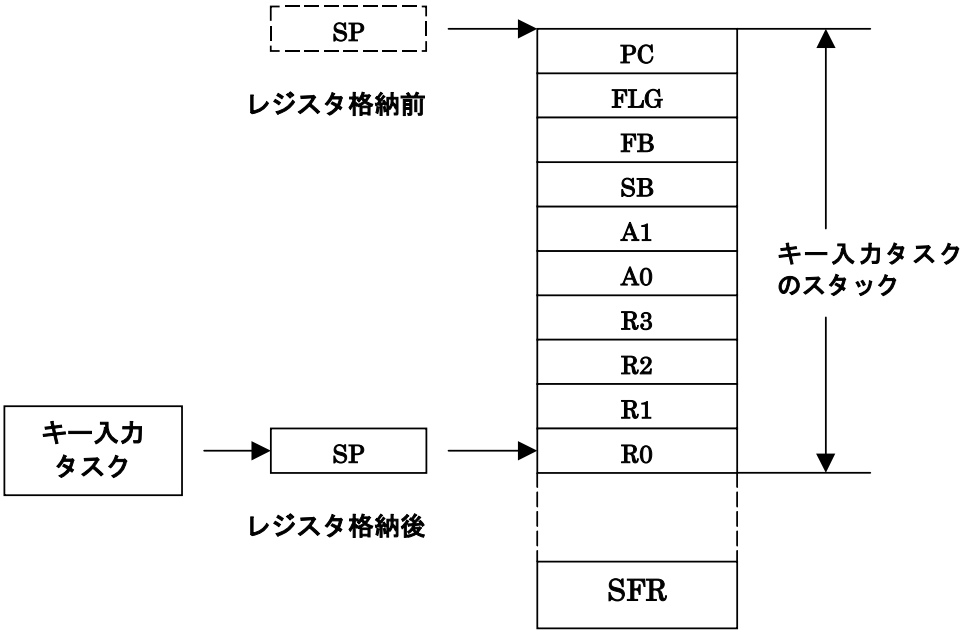


図 3.8　実際のレジスタとスタック領域の管理

3.2 サービスコール

リアルタイム OS をプログラマはプログラム中でどのように使用するのでしょうか?

これにはリアルタイム OS の機能をプログラムから何らかの形で呼び出すことが必要です。このリアルタイム OS の機能を呼び出すことをサービスコールといいます。すなわちサービスコールにより、タスクの起動などの処理を行なうことができます (図 3.9参照)。

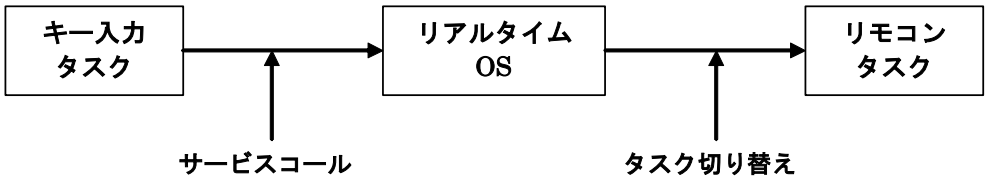


図 3.9　サービスコール

このサービスコールは、以下のように C 言語で応用プログラムを記述する場合は関数呼び出しで実現します。

```
sta_tsk(ID_main,3);
```

またアセンブリ言語で応用プログラムを記述する場合は以下のようにアセンブラマクロ呼び出しにより実現します。

```
sta_tsk #ID_main,#3
```

3.2.1 サービスコール処理

サービスコールが発行されると以下の手順により処理がおこなわれます。[8]

1. 現レジスタ内容を退避します。
2. スタックポインタをタスクのものからリアルタイム OS(システム)のものへ切り替えます。
3. サービスコール要求にしたがった処理を行います。
4. 次に実行するタスクの選択をおこないます。
5. スタックポインタをタスクのものに切り替えます。
6. レジスタ内容を復帰してタスクの実行を再開します。

サービスコールが発生してからタスク切り替えまでの処理の流れを図 3.10に示します。

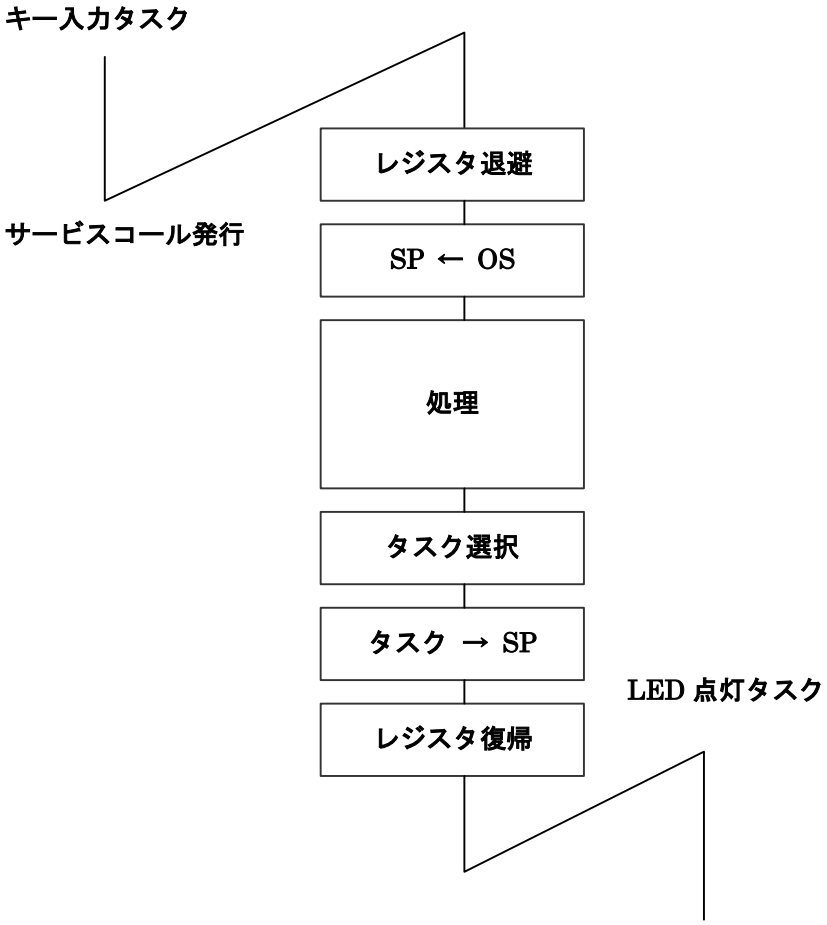


図 3.10 サービスコールの処理の流れ

[8] タスク切り替えの発生しないサービスコールはこの限りではありません。

3.2.2 サービスコールにおけるタスクの指定方法

各タスクの識別は、MR308 の内部では ID 番号でおこないます。
すなわち、"タスク ID 番号 1 のタスクを起動する"などというように管理されています。
しかし、プログラム中にタスクの番号を直接書き込むと非常に可読性の低いプログラムになってしまいます。たとえば、

```
sta_tsk(2,1);
```

とプログラム中に記述するとプログラマは絶えず ID 番号の 2 番のタスクは何かを知っている必要があります。また、他人がこのプログラムを見たときに ID 番号の 2 番のタスクが何かが一目では分かりません。
そこで MR308 ではタスクの識別をそのタスクの名前（関数もしくはシンボルの名前）で指定し、その名前からタスクの ID 番号への変換を MR308 に付属しているプログラム"コンフィギュレータ cfg308"が自動的におこないます。図 3.11にその様子を示します。

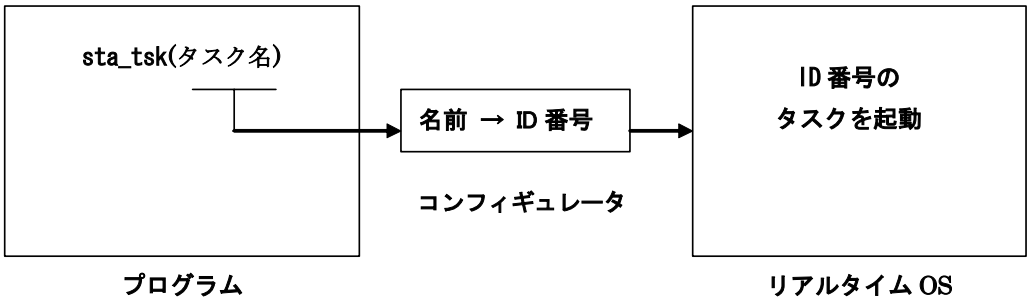


図 3.11 タスクの識別

これによりタスクの指定を以下のようにおこなえます。

```
sta_tsk(ID_task,1);
```

この例では、”ID_task”に対応するタスクを起動するように指定しています。
なお、タスクの名前から ID 番号への変換は、プログラムを生成するときにおこないます。したがって、この機能による処理速度の低下はありません。

3.3 タスク

本節ではタスクを MR308 がどのように管理しているかを説明します。

3.3.1 タスクの状態

リアルタイム OS ではタスクを実行するべきか否かを、タスクの状態を管理することにより制御しています。例えば、図 3.12にキー入力タスクの実行制御と状態の関係を示します。キー入力が発生した場合はそのタスクを実行しなければなりません。すなわち、キー入力タスクが実行状態となります。またキー入力を待っているときはタスクを実行する必要はありません。すなわち、キー入力タスクは待ち状態になっています。

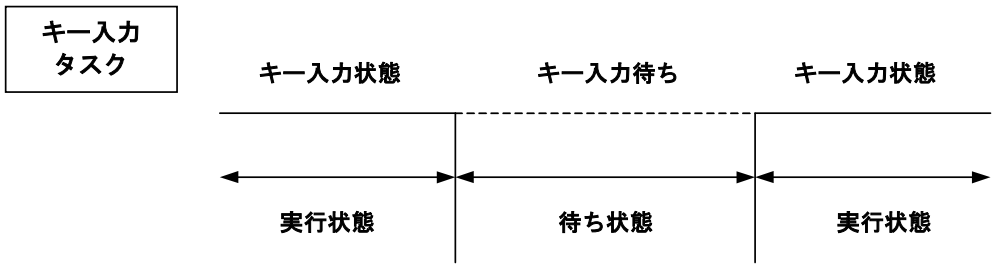


図 3.12 タスクの状態

MR308 では実行状態、待ち状態を含め以下の 6 つの状態を管理しています。

1. 実行状態（RUNNING 状態）
2. 実行可能状態（READY 状態）
3. 待ち状態（WAITING 状態）
4. 強制待ち状態（SUSPENDED 状態）
5. 二重待ち状態（WAITING-SUSPENDED 状態）
6. 休止状態（DORMANT 状態）

タスクは上記の 6 つの状態を遷移していきます。図 3.13に、タスクの状態遷移図を示します。

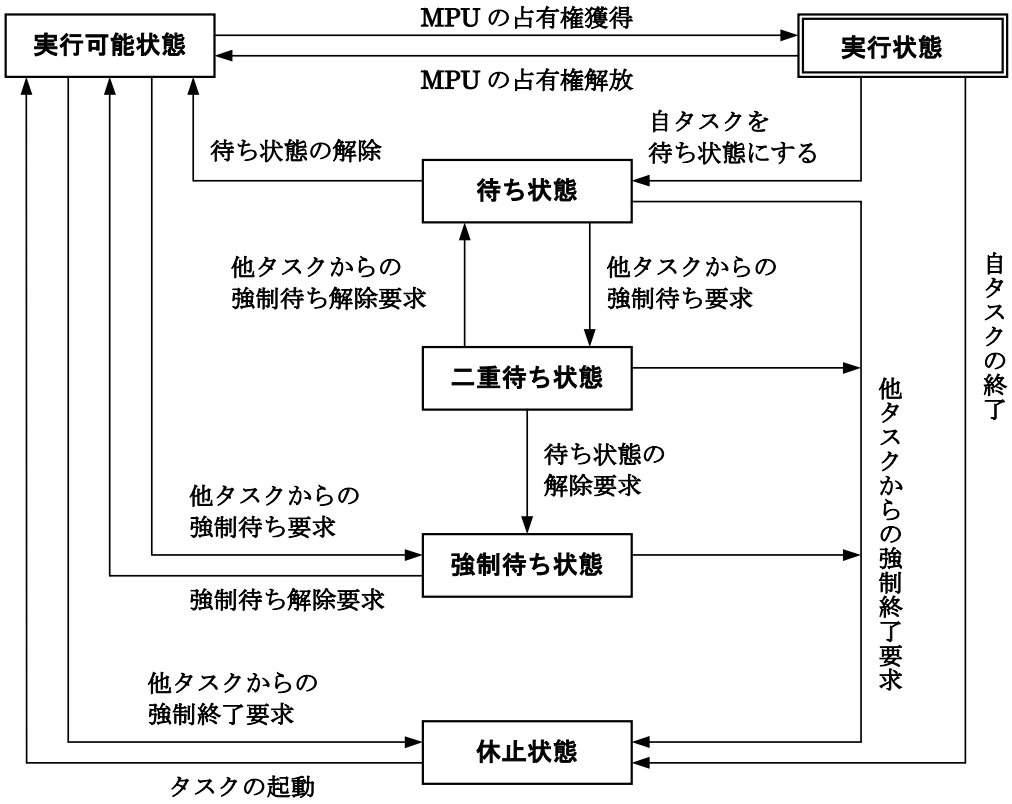


図 3.13 MR308 のタスク状態遷移図

1. 実行状態（RUNNING 状態）

タスクが、まさに現在実行中の状態を実行状態といいます。マイクロコンピュータは 1 つしかないのですから当然実行状態にあるのは常に 1 つだけです。

現在実行状態のタスクが他の状態に移行するには、以下の事象のうちどれかが発生した場合です。

- 自分で自タスクを正常終了させた場合[9]
- 自分で待ち状態に入った場合[10]
- 割り込み等の事象の発生により、その割り込みハンドラによって自タスクより優先度の高いタスクが実行可能状態になった場合
- 自タスクの優先度を変更することにより他の実行可能状態のタスクが自タスクより優先度が高くなった場合[11]
- 割り込み等の事象の発生により自タスクもしくは他の実行可能状態のタスクの優先度が変更され、そのため他の実行可能状態のタスクが自タスクより優先度が高くなった場合[12]
- rot_rdq,irot_rdq サービスコールにより、自タスク優先度のレディキューを回転し、実行権を放棄した場合

上記の事象が発生すると再スケジュールされて実行状態と実行可能状態にあるタスクのなかで最も優先度の高いタスクが実行状態に移され、そのタスクのプログラムが実行されます。

2. 実行可能状態（READY 状態）

タスクが実行される条件は整っているが、そのタスクより優先度の高いタスクもしくは同一優先度のタスクが実行されているために実行できずに実行待ち状態になっている状態を実行可能状態といいます。

[9] ext_tsk サービスコールによる

[10] dly_tsk, slp_tsk, tslp_tsk, wai_flg, twai_flg, wai_sem, twai_sem, rcv_mbx, trcv_mbx,snd_dtq,tsnd_dtq,rcv_dtq, trcv_dtq, vtsnd_dtq, vsnd_dtq,vtrcv_dtq,vrcv_dtq, get_mpf, tget_mpf サービスコールによる

[11] chg_pri サービスコールによる

[12] ichg_pri サービスコールによる

実行可能状態であるタスクで、レディキュー[13]では 2 番目に実行される可能性のあるタスクが実行状態になるのは、以下の事象の内いずれかが発生した場合です。

- 実行状態のタスクが自分で正常終了した場合[14]
- 実行状態のタスクが自分で待ち状態に入った場合[15]
- 実行状態のタスクが自分で優先度を変更することにより実行可能状態のタスクが実行状態のタスクより優先度が高くなった場合[16]
- 割り込み等の事象の発生により実行状態のタスクの優先度が変更され、そのため実行可能状態のタスクが実行状態のタスクより優先度が高くなった場合[17]
- rot_rdq,irot_rdq サービスコールにより、レディキュー先頭タスクが、自タスク優先度のレディキューを回転し、実行権を放棄した場合

3. 待ち状態（WAITING 状態）

実行状態のタスクが自分自身を待ち状態に移行させる要求を出すことにより、タスクは実行状態から待ち状態に移行することができます。待ち状態は通常入出力装置の入出力動作完了待ちや他のタスクの処理待ちなどの状態として使用されます。

実行待ち状態に移行するには以下の方法があります。

- slp_tsk サービスコールにより単純に待ち状態に移行します。この場合、他のタスクから明示的に待ち状態から解除されないと実行可能状態に移行しません。
- dly_tsk サービスコールにより一定時間待ち状態に移行します。この場合、指定時間経過するかもしくは他のタスクから明示的に待ち状態を解除することにより実行可能状態に移行します。
- wai_flg、wai_sem、rcv_mbx、snd_dtq、rcv_dtq、vsnd_dtq、vrcv_dtq、get_mpf サービスコールにより要求待ちで待ち状態に移行します。この場合、要求事項が満たされるかもしくは他のタスクから明示的に待ち状態を解除することにより実行可能状態に移行します。
- tslp_tsk、twai_flg、twai_sem、trcv_mbx、tsnd_dtq、trcv_dtq、vtsnd_dtq、vtrcv_dtq、tget_mpf サービスコールは、slp_tsk、wai_flg、wai_sem、rcv_mbx、tsnd_dtq、trcv_dtq、vsnd_dtq、vrcv_dtq、get_mpf サービスコールにタイムアウトを指定したサービスコールです。各サービスコールの要求待ちで待ち状態に移行します。この場合、要求事項が満たされるかもしくは、指定時間が経過した場合、実行可能状態に移行します。
- タスクが wai_flg、wai_sem、rcv_mbx、snd_dtq、rcv_dtq、vsnd_dtq、vrcv_dtq、get_mpf、tslp_tsk、twai_flg、twai_sem、trcv_mbx、tsnd_dtq、trcv_dtq、vtsnd_dtq、vtrcv_dtq、tget_mpf サービスコールにより要求待ちで待ち状態になると、その要求事項により次の待ち行列のいずれかにつながります。

- イベントフラグ待ち行列
- セマフォ待ち行列
- メールボックスメッセージ受信待ち行列
- データキューデータ送信待ち行列
- データキューデータ受信待ち行列
- short データキューデータ送信待ち行列
- short データキューデータ受信待ち行列
- 固定長メモリプールメモリ獲得待ち行列

[13] レディキューについては次節参照
[14] ext_tsk サービスコールによる
[15] dly_tsk, slp_tsk, tslp_tsk, wai_fg, twai_flg, wai_sem, twai_sem, rcv_mbx, trcv_mbx, snd_dtq, tsnd_dtq, rcv_dtq, trcv_dtq, get_mpf,tget_mpf, vsnd_dtq, vtsnd_dtq, vrcv_dtq, vtrcv_dtq サービスコールによる
[16] chg_pri サービスコールによる
[17] icg_pri サービスコールによる

4. 強制待ち状態（SUSPENDED 状態）

実行状態のタスクから sus_tsk サービスコールが発行される、もしくはハンドラから isus_tsk サービスコールが発行されると、サービスコールにより指定された実行可能なタスクもしくは実行中のタスクは強制待ち状態になります。なお待ち状態のタスクが指定された場合は二重待ち状態になります。

強制待ち状態は入出力エラー等の発生により実行可能なタスクもしくは実行中のタスク[18]が処理を一時的に中断させるためにスケジューリングから外された状態です。すなわち実行可能状態のタスクに対して強制待ち要求が出された場合、そのタスクは実行待ち行列から外されます。

なお、強制待ち要求のキューイングは行いません。したがって強制待ち要求は実行状態、実行可能状態、待ち状態[19]にあるタスクにのみ行えます。すでに強制待ち状態にあるタスクに強制待ち要求した場合には、エラーコードが返されます。

5. 二重待ち状態（WAITING-SUSPENDED 状態）

待ち状態にあるタスクに強制待ちの要求が出された場合、タスクは二重待ち状態になります。wai_flg、wai_sem、rcv_mbx、snd_dtq、rcv_dtq、vsnd_dtq、vrcv_dtq、get_mpf、tslp_tsk、twai_flg、twai_sem、trcv_mbx、tsnd_dtq、trcv_dtq、vtsnd_dtq、vtrcv_dtq、tget_mpf サービスコールによる要求待ちで待ち状態にあるタスクに対して強制待ち要求が出された場合、そのタスクは二重待ち状態に移行します。

また、二重待ち状態のタスクは待ち条件が解除されると強制待ち状態になります。待ち条件が解除されるには以下の場合が考えられます。

- wup_tsk、iwup_tsk サービスコールにより起床する場合
- dly_tsk、tslp_tsk サービスコールにより待ち状態になったタスクが時間経過により起床される場合
- wai_flg、wai_sem、rcv_mbx、snd_dtq、rcv_dtq、vsnd_dtq、vrcv_dtq、get_mpf、tslp_tsk、twai_flg、twai_sem、trcv_mbx、tsnd_dtq、trcv_dtq、vtsnd_dtq、vtrcv_dtq、tget_mpf サービスコールにより待ち状態になったタスクの要求が満たされた、もしくは、指定時間が経過した場合
- rel_wai、irel_wai サービスコールにより待ち状態が強制解除される場合

二重待ち状態のタスクに強制待ち解除要求[20]がだされると待ち状態になります。なお、強制待ち状態にあるタスクが自分自身を待ち状態にする要求は出せないため、強制待ち状態から二重待ち状態への移行は発生しません。

6. 休止状態（DORMANT 状態）

通常は、MR308 システムに登録されているが起動していない状態です。この状態になるには以下の 2 つの場合があります。

- タスクが起動をかけられるのを待っている場合
- タスクが正常終了[21]もしくは強制終了[22]により終了した場合

---

[18] ハンドラから isus_tsk サービスコールにより実行タスクを強制待ち状態にする場合は、実行状態から直接強制待ち状態に移行されます。例外的にこの場合のみ実行状態から強制待ち状態に移行する場合はあることに注意してください。

[19] 待ち状態にあるタスクに対して強制待ち要求をおこなうと二重待ち状態になります。

[20] rsm_tsk, irsm_tsk サービスコール

[21] ext_tsk サービスコール

[22] ter_tsk サービスコール

3.3.2 タスクの優先度とレディキュー

リアルタイム OS では実行したいタスクが同時にいくつも発生することがあります。

このときにどのタスクを実行するかを判断することが必要になります。そこでタスクに実行の優先度をつけ、優先度の高いタスクから実行するようにします。すなわち、処理を素早くおこなう必要のあるタスクの優先度を高くしておけば実行したいときに素早く実行することができるようになります。

MR308 では同一の優先度を複数のタスクに与えることができます。そこで、実行可能になったタスクの実行順を制御するためにタスクの待ち行列 (レディキュー)を生成します。

図 3.14[23]にレディキューの構造を示します。レディキューは優先度ごとに管理され、タスクが接続されている最も優先度の高い待ち行列の先頭タスクを実行状態にします。[24]

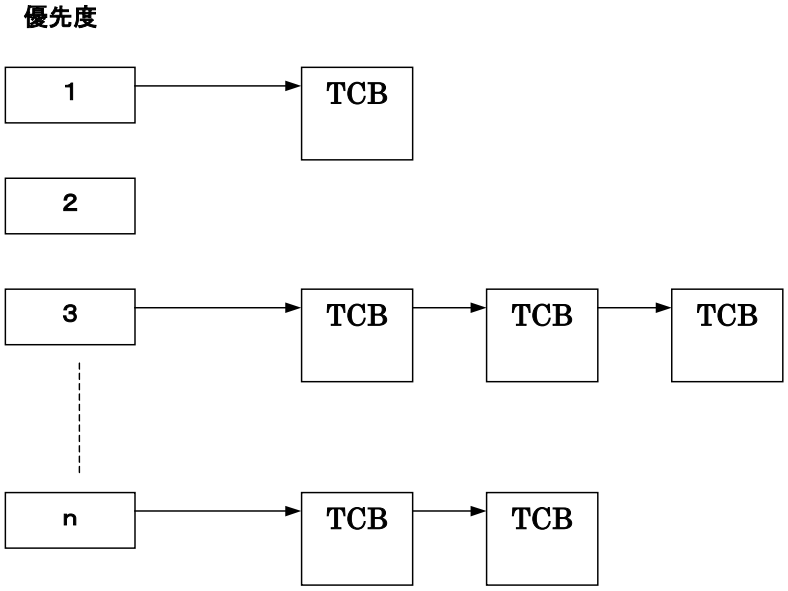


図 3.14 レディキュー（実行待ち状態）

[23] TCB:タスクコントロールブロックについては次節で述べます。

[24] 実行状態のタスクはレディキューにつながれたままです。

3.3.3 タスクの優先度と待ち行列

μITRON 4.0 仕様のスタンダードプロファイルでは、各オブジェクトの待ち方にタスクの優先度順に待ち行列をつなぐ(TA_TPRI 属性)、FIFO 順に待ち行列につなぐ(TA_TFIFO)の２種類をサポートすることになっています。MR308 でもこの２種類の待ち方をサポートしています。

図 3.15、図 3.16 にタスクが、”taskD”、”taskC”、”taskA”、”taskB”の順で待ち行列につながれたときの様子を示します。

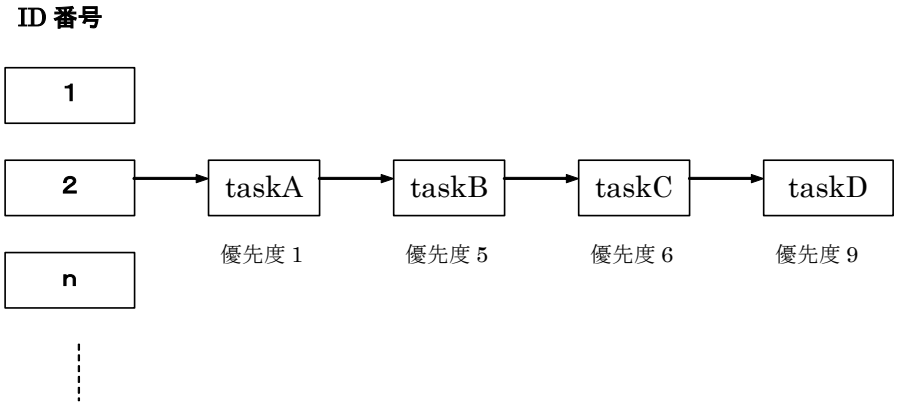


図 3.15 TA_TPRI 属性の待ち行列

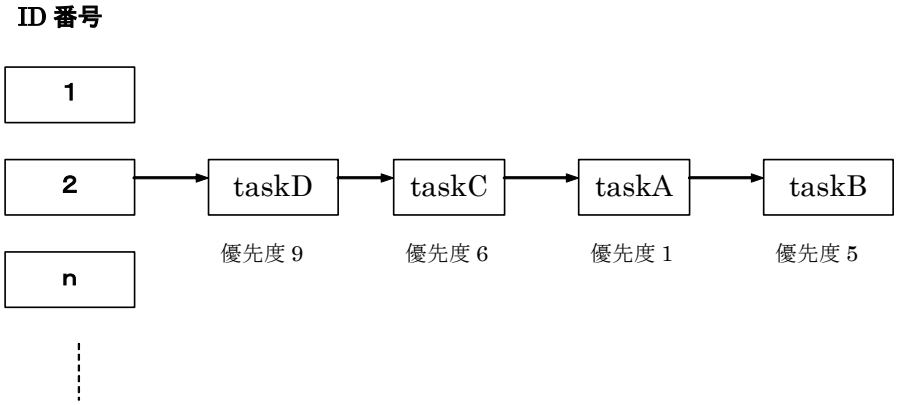


図 3.16 TA_TFIFO 属性の待ち行列

3.3.4 タスクコントロールブロック (TCB)

タスクコントロールブロック (TCB)とは、リアルタイム OS がそれぞれのタスクの状態や優先度などを管理するデータブロックのことを言います。

MR308 ではタスクの以下の情報をタスクコントロールブロックとして管理しています。

- タスク接続ポインタ
  レディキューなどを構成するときに使用するタスク接続用ポインタ
- タスクの状態
- タスクの優先度
- タスクのレジスタ情報など[25]を格納したスタック領域のポインタ (現在の SP レジスタの値)
- 起床要求カウンタ
  タスクの起床要求カウンタを蓄積する領域
- タイムアウトカウンタ、待ちフラグパターン
  タスクがタイムアウト待ち状態である時は、残りの待ち時間が格納され、フラグ待ち状態であれば、フラグの待ちパターンがこの領域に格納されます。
- フラグ待ちモード
  イベントフラグ待ちの時の待ちモード
- タイマキュー接続ポインタ
  タイムアウト機能を使用した場合に使用する領域です。タイマキューを構成する時に使用するタスクの接続用ポインタを格納する領域です。
- フラグ待ちパターン
  タイムアウト機能を使用した場合に使用する領域です。
  タイムアウト機能付きのイベントフラグ待ちのサービスコール (twai_flg)を使用した場合に、フラグ待ちパターンが格納されます。なお、この領域は、イベントフラグを使用しない場合は、確保されません。
- 起動要求カウンタ
  タスクの起動要求を蓄積する領域
- タスクの拡張情報
  タスク生成時に設定する、タスクの拡張情報がこの領域に格納されます。

タスクコントロールブロックを図 3.17に示します。

[25] これをタスクコンテキストと呼びます。

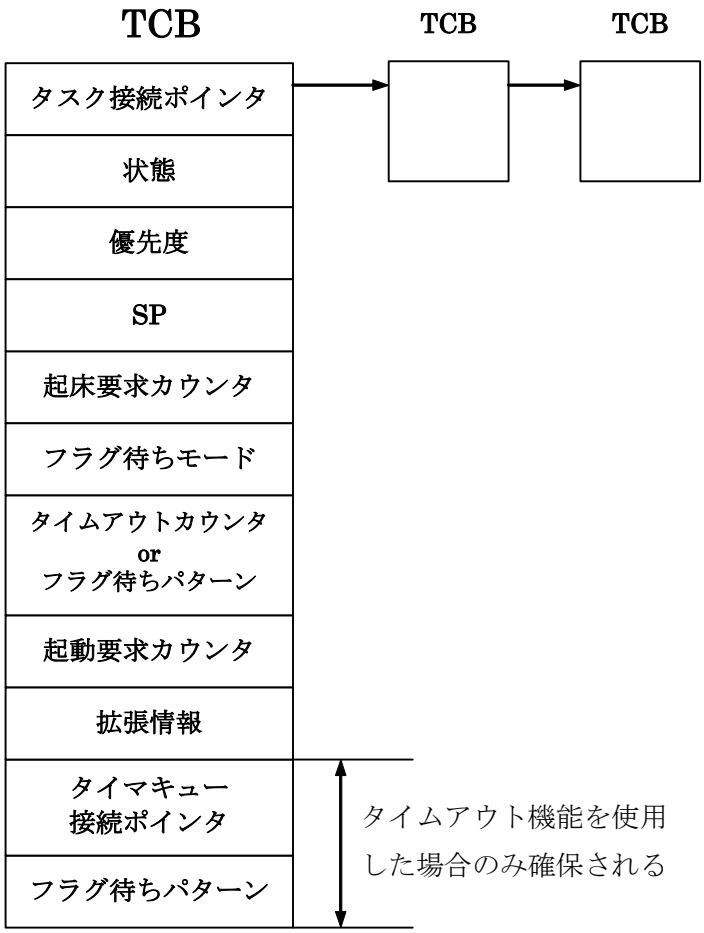


図 3.17 タスクコントロールブロック

3.4 システムの状態

3.4.1 タスクコンテキストと非タスクコンテキスト

システムは、「タスクコンテキスト」か「非タスクコンテキスト」のいずれかのコンテキスト状態で実行します。タスクコンテキストと非タスクコンテキストの違いを表 3-1 タスクコンテキストと非タスクコンテキストに示します。

表 3-1 タスクコンテキストと非タスクコンテキスト

	タスクコンテキスト	非タスクコンテキスト
呼び出し可能なサービスコール	タスクコンテキストから呼び出せるもの	非タスクコンテキストから呼び出せるもの
タスクスケジューリング	キューの状態が変化し、ディスパッチ禁止状態、CPU ロック状態のいずれでもない場合に発生	発生しない
スタック	ユーザスタック	システムスタック

非タスクコンテキストで実行される処理には以下のものがあります。

割り込みハンドラ

ハードウェア割り込みにより起動されるプログラムを割り込みハンドラと呼びます。割り込みハンドラの起動には MR308 は全く関与しません。したがって割り込みハンドラの入り口アドレスを割り込みベクターテーブルに直接書き込みます。

割り込みハンドラには、カーネル管理外(OS 独立)割り込み、カーネル管理(OS 依存)割り込みの 2 種類があります。各割り込みについては、5.5節を参照して下さい。

システムクロック割り込みハンドラ(isig_tim)も割り込みハンドラに含まれます。

周期ハンドラ

周期ハンドラはあらかじめ設定された時間毎に周期的に起動されるプログラムです。 設定された周期ハンドラを無効にするか有効にするかは sta_cyc(ista_cyc)や stp_cyc(istp_cyc)サービスコールによりおこないます。

また、周期ハンドラ起動時刻は、set_tim(iset_tim)による、時刻変化の影響を受けません。

アラームハンドラ

アラームハンドラは、 指定した相対時刻経過後に起動されるハンドラです。起動時刻は、sta_alm(ista_alm)設定時の時刻に対する相対時刻で決定され、set_tim(iset_tim)による、時刻変化の影響を受けません。

周期ハンドラとアラームハンドラはシステムクロック割り込み(タイマ割り込み)ハンドラからサブルーチンコールで呼び出されます(図 3.18 周期ハンドラ、アラームハンドラの起動参照)。したがって、周期ハンドラ、アラームハンドラはシステムクロック割り込みハンドラの一部として動作します。なお、周期ハンドラ、アラームハンドラが呼び出されるときは、システムクロック割り込みの割り込み優先レベルの状態で実行されます。

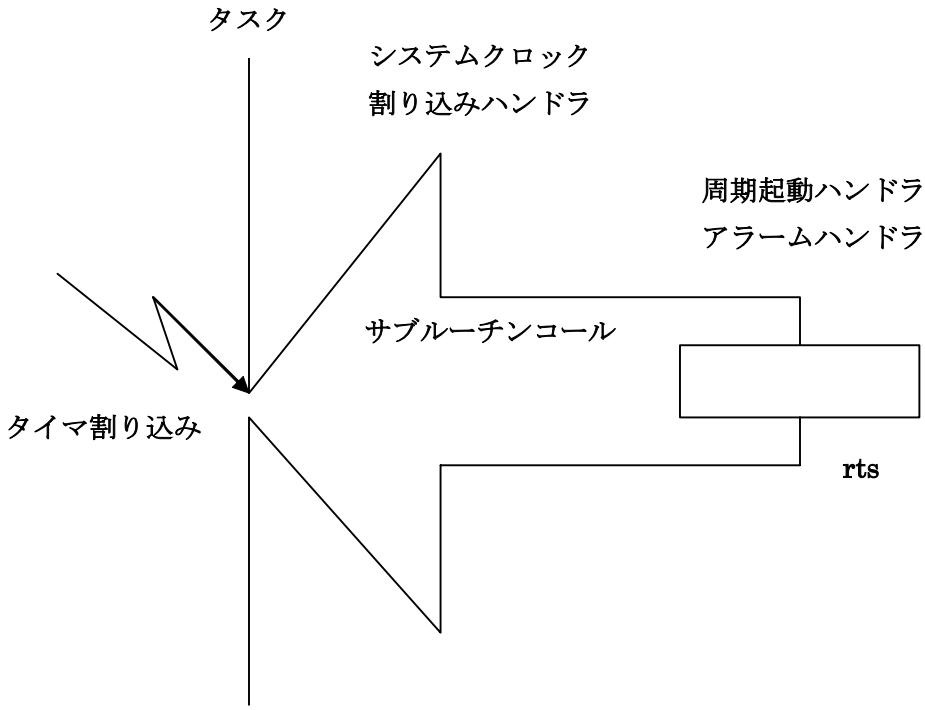


図 3.18 周期ハンドラ、アラームハンドラの起動

3.4.2 ディスパッチ禁止/許可状態

システムは、ディスパッチ許可状態、ディスパッチ禁止状態のいずれかの状態をとります。ディスパッチ禁止状態では、タスクスケジューリングが行われません。また、サービスコール発行タスクが待ち状態に移行するようなサービスコールも呼び出すことはできません。[26]

ディスパッチ禁止状態へは、dis_dsp サービスコール、ディスパッチ許可状態へは ena_dsp サービスコールの発行により遷移することができます。また、sns_dsp サービスコールによりディスパッチ禁止状態がどうか知ることができます。

[26] MR308 は、ディスパッチ禁止状態から発行できないサービスコールを発行してもエラーは返しませんが、その場合の動作は保証しません。

3.4.3 CPU ロック/ロック解除状態

システムは、CPU ロック状態か CPU ロック解除状態のいずれかの状態をとります。CPU ロック状態では、すべての外部割り込みの受付が禁止され、タスクスケジューリングも行われません。

CPU ロック状態へは loc_cpu(iloc_cpu)サービスコール、CPU ロック解除状態へは unl_cpu(iunl_cpu)サービスコール発行により遷移します。また、sns_loc サービスコールによって CPU ロック状態かどうか調べることができます。

CPU ロック状態から発行できるサービスコールは表 3-2のように制限されます。[27]

表 3-2 CPU ロック状態で使用可能なサービスコール

loc_cpu	iloc_cpu	unl_cpu	iunl_cpu
ext_tsk	sns_ctx	sns_loc	sns_dsp
sns_dpn			

3.4.4 ディスパッチ禁止状態と CPU ロック状態

μITRON 4.0 仕様では、ディスパッチ禁止状態と CPU ロック状態が明確に区別されるようになりました。従って、ディスパッチ禁止状態で unl_cpu サービスコールを発行したとしても、ディスパッチ禁止状態のまま変化せず、タスクスケジューリングは行われません。状態遷移をまとめると表 3-3のようになります。

表 3-3 dis_dsp,loc_cpu に関する CPU ロック、ディスパッチ禁止状態遷移

状態番号	状態の内容		dis_dsp を実行	ena_dsp を実行	loc_cpu を実行	unl_cpu を実行
	CPU ロック状態	ディスパッチ禁止状態				
1	○	×	×	×	→ 1	→ 3
2	○	○	×	×	→ 2	→ 4
3	×	×	→ 4	→ 3	→ 1	→ 3
4	×	○	→ 4	→ 3	→ 2	→ 4

[27] MR308 は、CPU ロック状態から発行できないサービスコールを発行してもエラーは返しませんが、その場合の動作は保証しません。

3.5 MR308 カーネルの構成

3.5.1 モジュール構成

MR308 カーネルは、図 3.19に示すモジュールから構成されています。これらの個々のモジュールはそれぞれのモジュールの機能を実現する関数群より構成されています。

MR308 カーネルはライブラリ形式で提供されシステム生成時に必要な機能のみがリンクされます。すなわちこれらのモジュールを構成する関数群の中で使用している関数のみをリンケージエディタ LN308 の機能によりリンクします。ただし、スケジューラとタスク管理の一部および時間管理の一部は必須機能関数ですので常時リンクされます。

アプリケーションプログラムはユーザーが作成するプログラムで、タスク・割り込みハンドラ・アラームハンドラおよび周期ハンドラから構成されます。

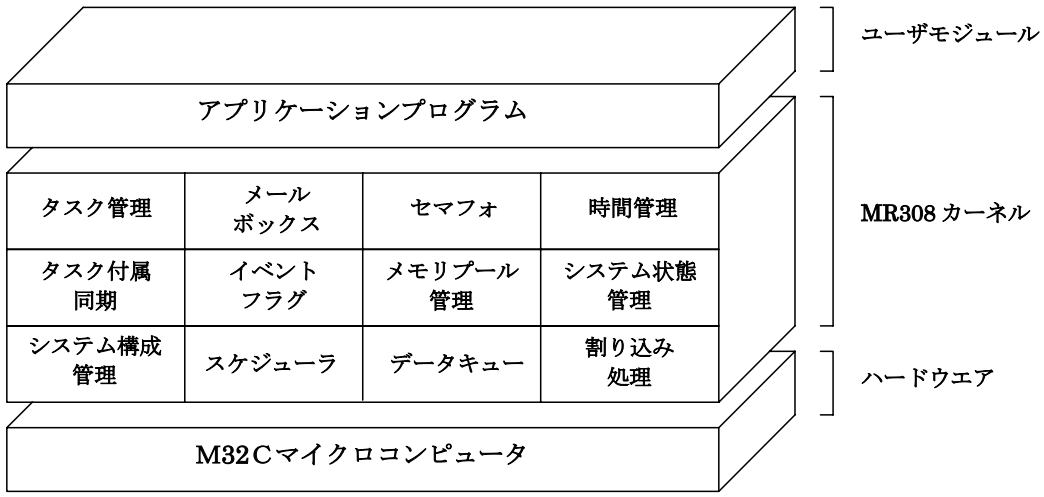


図 3.19 MR308 の構成

3.5.2 モジュール概要

MR308 カーネルを構成する各モジュールの概要を説明します。

- スケジューラ
  タスクの持つ優先度に基づいて、タスクの処理待ち行列を形成し、その待ち行列の先頭にある優先度の高い (優先度の値の小さい)タスクの処理を実行するよう制御をおこないます。

- タスク管理
  実行・実行可能・待ち・強制待ち等のタスク状態の管理をおこないます。

- タスク付属同期
  他タスクからタスクの状態を変化させることによりタスク間の同期をとります。

- 割り込み管理
  割り込みハンドラからの復帰処理をおこないます。

- 時間管理
  MR308 カーネルで使用するシステムタイマの設定、タイムアウトの処理、ユーザーの作成したアラームハンドラ[28]、周期ハンドラ[29]の起動をおこないます。

- システム状態管理
  MR308 のシステム状態を取得します。

- システム構成管理
  MR308 カーネルのバージョン番号等の情報を報告します。

- 同期・通信
  タスク間の同期をとったり、タスク間の通信をおこなうための機能です。以下の 4 つの機能モジュールが用意されています。
  - ◆ イベントフラグ
    MR308 内部で管理されているフラグが立っているか否かによりタスクを実行するかしないかを制御します。これによりタスク間の同期をとることができます。
  - ◆ セマフォ
    MR308 内部で管理されているセマフォカウンタ値によりタスクを実行するかしないかを制御します。これによりタスク間の同期をとることができます。
  - ◆ メールボックス
    タスク間のデータの通信をデータの先頭アドレスを渡すことによりおこないます。
  - ◆ データキュー
    タスク間の 32 ビットデータの通信をおこないます。

- メモリプール管理
  タスクまたはハンドラが使用するメモリ領域の動的な獲得および解放を行います。

- 拡張機能機能
  μITRON 4.0 仕様の仕様外の機能で short データキュー、オブジェクトのリセット処理を行います。

[28] 指定時刻に一回のみ起動されるハンドラです。
[29] 周期的に起動されるハンドラです。

3.5.3 タスク管理機能

タスク管理機能とは、タスクの起動・終了・優先度の変更等のタスク操作をおこなう機能です。
MR308 カーネルが提供するタスク管理機能のサービスコールには、次のものがあります。

- タスクを起動する（act_tsk,iact_tsk）
  あるタスクから、他タスクを起動することにより、起動対象となるタスクの状態を休止状態から実行可能状態もしくは実行状態に移行します。
  本サービスコールでは、sta_tsk(ista_tsk)と違い、起動要求は蓄積しますが、起動コードを指定することはできません。

- タスクを起動する（sta_tsk,ista_tsk）
  あるタスクから、他タスクを起動することにより、起動対象となるタスクの状態を休止状態から実行可能状態もしくは実行状態に移行します。
  本サービスコールは、act_tsk(iact_tsk)と違い、起動要求は蓄積しませんが、起動コードを指定することができます。

- 自タスクを終了する（ext_tsk）
  自タスクを終了するとタスクの状態が休止状態になります。これにより再起動されるまで、このタスクは実行しません。起動要求が蓄積されている場合は、再度タスクの起動処理を行います。その際、自タスクは、リセットされたように振る舞います。
  C 言語で記述した場合、本サービスコールは、タスク終了時に明示的に記述されていなくても、タスクからリターンする際に自動的に呼び出されます。

- 他タスクを強制的に終了させる（ter_tsk）
  休止状態以外の他のタスクを強制的に終了させ休止状態にします。起動要求が蓄積されている場合は、再度タスクの起動処理を行います。その際、自タスクは、リセットされたように振る舞います。（図 3.20参照）

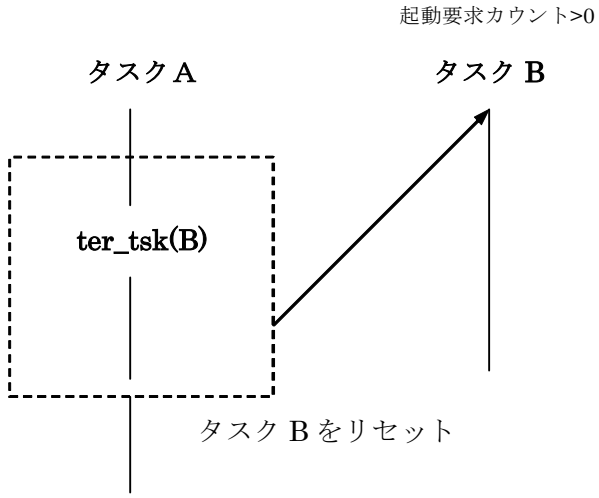
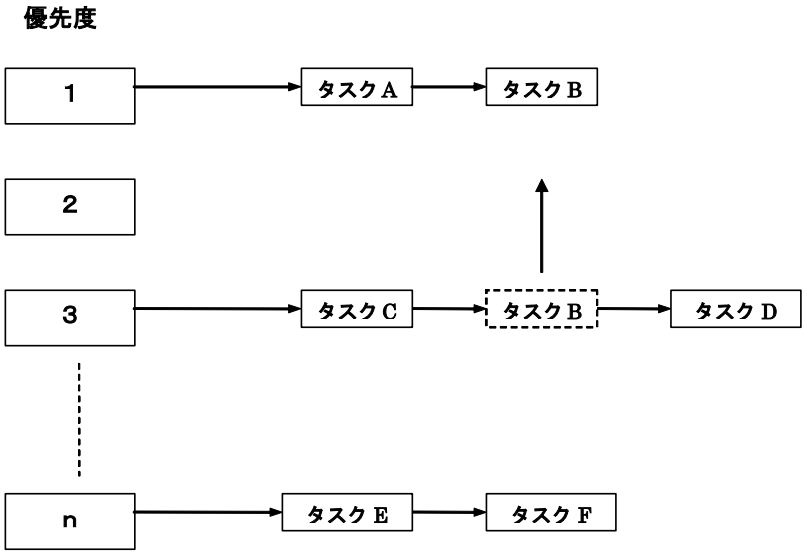


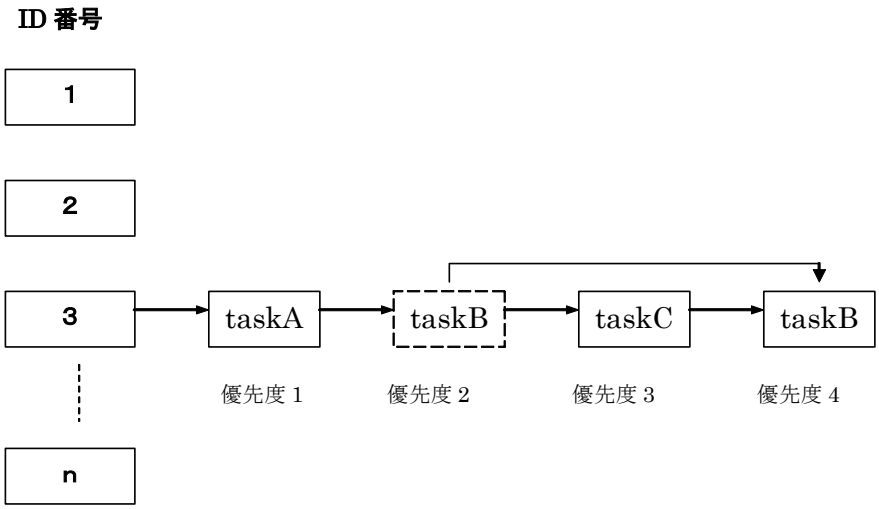
図 3.20 タスクのリセット

- タスクの優先度を変更する（chg_pri, ichg_pri）
  タスクの優先度を変更するとそのタスクが実行可能状態もしくは実行状態であるときは、レディキューも更新されます。（図 3.21参照）
  また、対象タスクが TA_TPRI 属性を持つオブジェクトの待ち行列につながれている場合は、待ち行列も更新されます。（図 3.22参照）



タスク B の優先度を 3 から 1 に変更した場合

図 3.21 優先度の変更



タスク B の優先度を 2 から 4 に変更した場合

図 3.22 待ちキューのつなぎ換え

- タスクの優先度を取得する (get_pri, iget_pri)
  タスクの優先度を取得します。
- タスクの状態を参照する(簡易版) (ref_tst,iref_tst)
  対象タスクの状態を参照します。
- タスクの状態を参照する (ref_tsk,iref_tsk)
  対象タスクの状態およびその優先度等を参照します。

3.5.4 タスク付属同期機能

タスク付属同期機能とは、タスク間の同期をとるためにタスクを待ち状態（もしくは強制待ち状態・二重待ち状態)にしたり、待ち状態になったタスクを起床させたりする機能です。

MR308 カーネルが提供するタスク付属同期サービスコールには次のものがあります。

- タスクを待ち状態に移行する（slp_tsk,tslp_tsk）
- 待ち状態のタスクを起床する（wup_tsk, iwup_tsk）

  slp_tsk、tslp_tsk サービスコールにより待ち状態に入ったタスクを起床させます。

  slp_tsk、tslp_tsk サービスコール以外の条件で待ち状態にあるタスクは起床できません。

  slp_tsk、tslp_tsk サービスコール以外の条件で待ちに入ったタスクや休止状態を除く他の状態[30]のタスクに対して wup_tsk、iwup_tsk サービスコールにより起床要求をおこなうと、この起床要求だけが蓄積されます。

  したがって、例えば実行状態のタスクに対して起床要求をおこなうと、この起床要求が一時的に記憶されます。そして、その実行状態のタスクが slp_tsk、tslp_tsk サービスコールにより待ち状態に入ろうとした時、蓄積された起床要求が有効になり、待ち状態にならずに再び実行を続けます。（図 3.23 参照)
- タスクの起床要求を無効にする（can_wup）

  蓄積された起床要求をクリアします。（図 3.24参照）

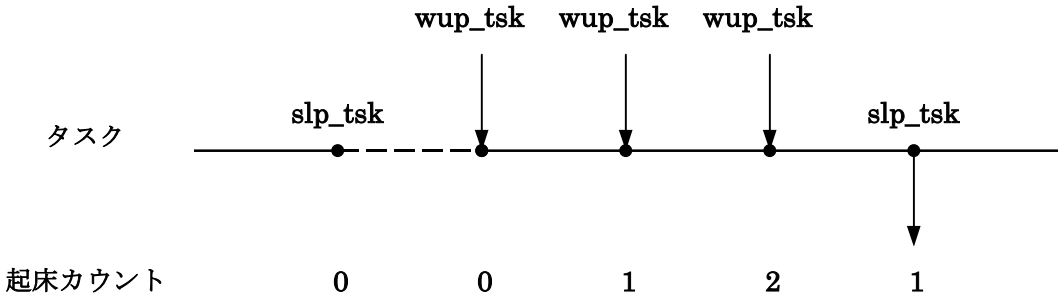


図 3.23 起床要求の蓄積

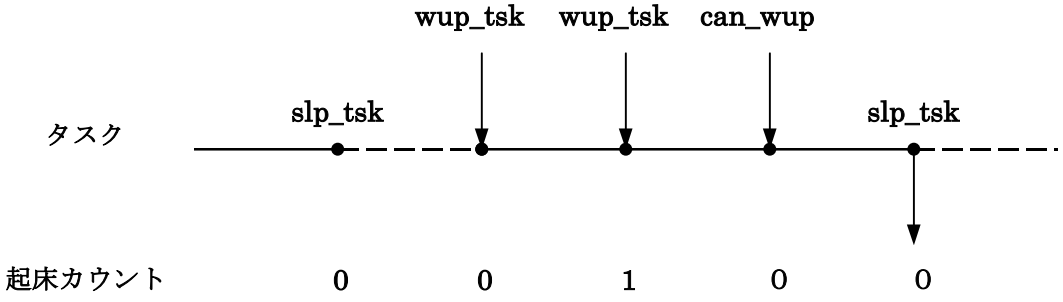


図 3.24 起床要求のキャンセル

[30] 待ち状態であっても以下の条件で待っているタスクは起床されませんのでご注意ください。
イベントフラグ待ち状態、セマフォ待ち状態、データ送信待ち状態、データ受信待ち状態、タイムアウト待ち状態、固定長メモリプール獲得待ち、short データ送信待ち、short データ受信待ち

- タスクを強制待ち状態に移行する（sus_tsk, isus_tsk）

- 強制待ち状態のタスクを再開する（rsm_tsk, irsm_tsk）
  タスクの実行を強制的に待たせたり、実行を再開したりします。実行可能状態のタスクを強制待ちすれば強制待ち状態になり、待ち状態のタスクを強制待ちすれば二重待ち状態になります。MR308 では最大強制待ち要求ネスト数は 1 であるため、強制待ち状態のタスクに sus_tsk を発行するとエラー E_QOVR が返されます。(図 3.25参照)

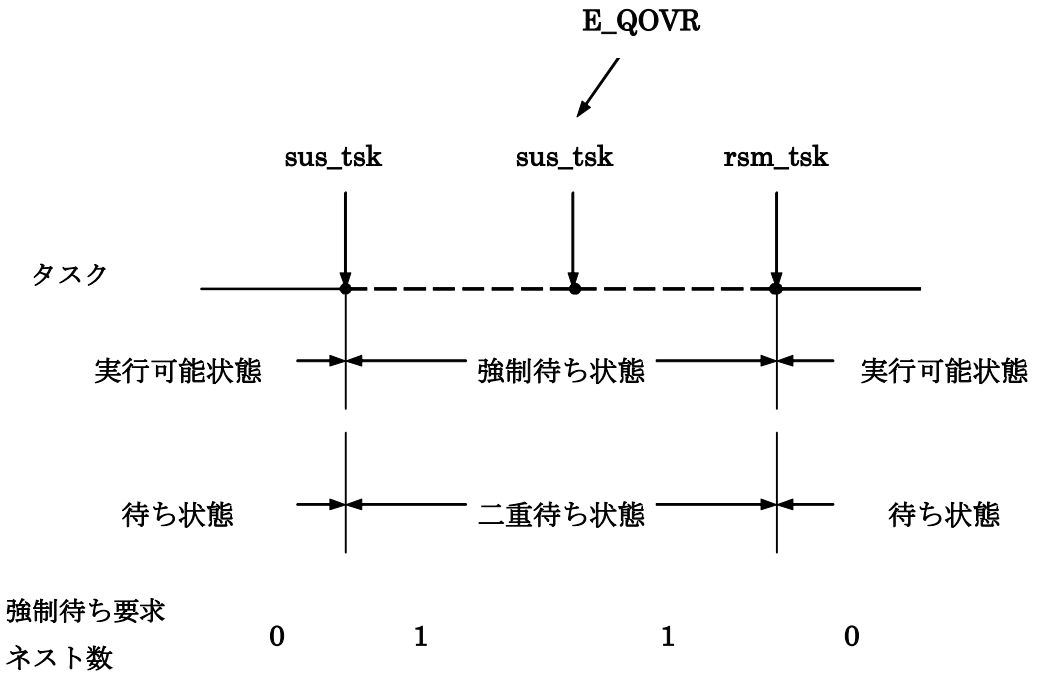


図 3.25 タスクの強制待ちと再開

- 強制待ち状態のタスクを強制再開する（frsm_tsk, ifrsm_tsk）
  強制待ち要求のネスト数をすべてクリアし、タスクの実行を強制的に再開します。MR308 では最大強制待ち要求ネスト数は 1 であるため、rsm_tsk,irsm_tsk と同じ動作となります。(図 3.26参照)

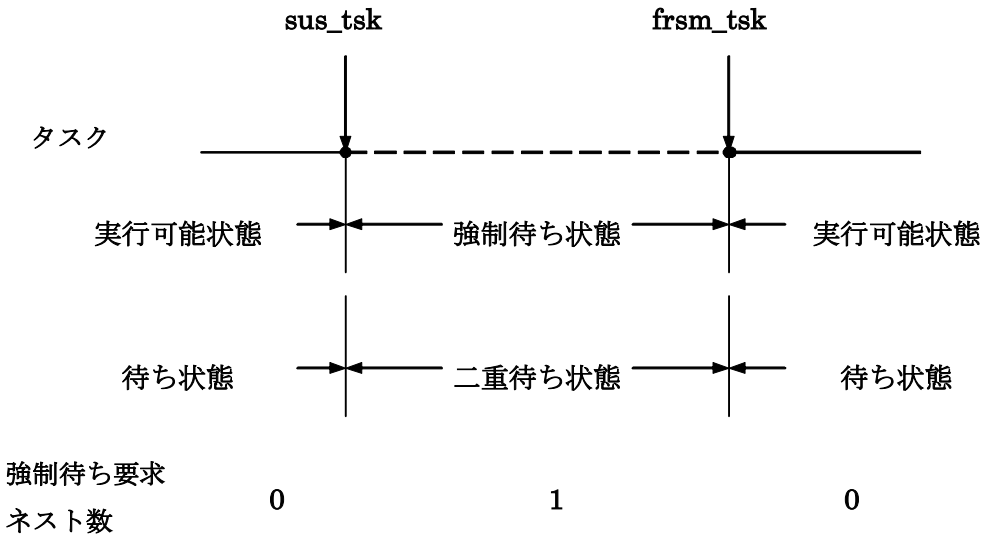


図 3.26 タスクの強制待ちと強制再開

- タスクの待ち状態を強制解除する（rel_wai, irel_wai）
  タスクの待ち状態を強制的に解除します。解除される待ち状態は以下の条件により待ちに入ったタスクです。
  - タイムアウト待ち状態
  - slp_tsk サービスコールによる（+タイムアウト有)待ち状態
  - イベントフラグ（+タイムアウト有)待ち状態
  - セマフォ（+タイムアウト有)待ち状態
  - メッセージ（+タイムアウト有)待ち状態
  - データ送信（+タイムアウト有)待ち状態
  - データ受信（+タイムアウト有)待ち状態
  - 固定長メモリブロック（+タイムアウト有)獲得待ち状態
  - short データ送信（+タイムアウト有)待ち状態
  - short データ受信（+タイムアウト有)待ち状態

- タスクを一定時間待ち状態に移行します（dly_tsk）
  タスクを一定時間待たせます。図 3.27に dly_tsk サービスコールにより 10msec 間タスクの実行を待たせる例を示します。タイムアウト値として指定する単位は、タイムティック数ではなく ms 単位で指定します。

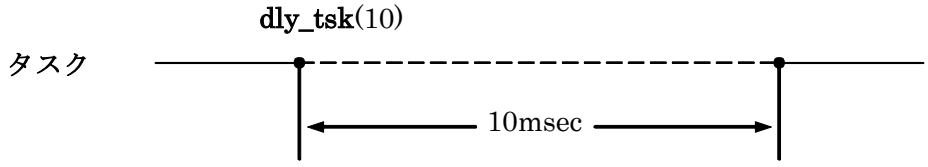


図 3.27 dly_tsk サービスコール

3.5.5 同期・通信機能 (セマフォ)

セマフォは複数のタスクで共有する装置などの資源の競合を防ぐための機能です。例えば図 3.28に示すような場合、すなわち通信回線が３本しかないシステムに４つのタスクが回線を獲得しようと競合した場合に、通信回線を競合することなくタスクに接続することがセマフォを用いるとできます。

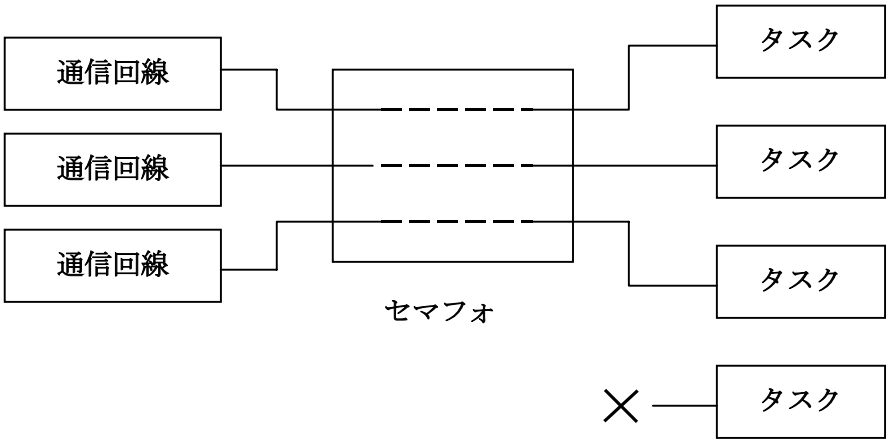


図 3.28 セマフォによる排他制御

セマフォは内部にセマフォカウンタと呼ばれる計数値を持っており、そのセマフォカウンタに基づきセマフォを獲得・解放をおこなうことによって資源の競合を防ぎます。(図 3.29参照)

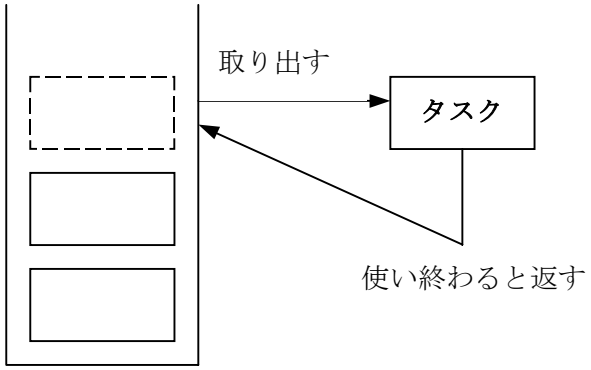


図 3.29 セマフォカウンタ

MR308 カーネルが提供するセマフォ同期のサービスコールには次のものがあります。

- セマフォに対する信号操作 (sig_sem, isig_sem)
  セマフォに信号をおくります。すなわち、セマフォを待っているタスクがあればそのタスクを起床し、なければセマフォカウンタを１増やします。

- セマフォ獲得操作 (wai_sem, twai_sem)
  セマフォを待ちます。セマフォカウンタが０であればセマフォを得ることができませんので待ち状態になります。

- セマフォ獲得操作 (pol_sem,ipol_sem)
  セマフォを得ます。得るべきセマフォがなければ待ち状態に入らずにエラーコードをかえします。

- セマフォの状態を参照する (ref_sem,iref_sem)
  対象セマフォの状態を参照します。対象セマフォのカウント値や待ちタスクの有無を参照します。

wai_sem、sig_sem サービスコールを用いたタスクの実行制御の例を図 3.30に示します。

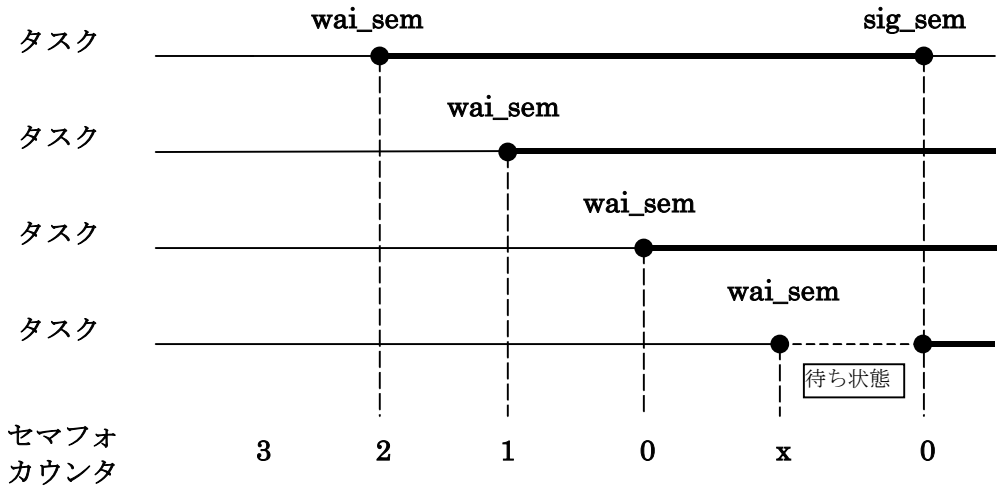


図 3.30 セマフォによるタスクの実行制御

3.5.6 同期・通信機能 (イベントフラグ)

イベントフラグは複数のタスクの実行の同期をとるための MR308 内部に持つ機構です。イベントフラグは、フラグ待ちパターンと 16 ビットのビットパターンによりタスクの実行制御をおこないます。タスクは、設定したフラグ待ちの条件が満たされるまで待ちます。

ひとつのイベントフラグの待ち行列に複数の待ちタスクの接続を許可するかどうかをイベントフラグ属性 TA_WSGL、TA_WMUL を指定することにより決定することができます。

また、イベントフラグ属性に TA_CLR を指定することにより、イベントフラグが待ち条件を満たした場合イベントフラグのビットパターンをすべてクリアすることができます。

MR308 カーネルが提供するイベントフラグのサービスコールには次のものがあります。

- イベントフラグをセットする（set_flg, iset_flg)
  イベントフラグをセットします。これにより、このイベントフラグの待ちパターンを待っていたタスクは待ち解除されます。

- イベントフラグをクリアする（clr_flg,iclr_flg)
  イベントフラグをクリアします。

- イベントフラグを待つ（wai_flg, twai_flg)
  イベントフラグがあるパターンにセットされるのを待ちます。イベントフラグを待つ時のモードは、以下に示す 2 類があります。
  - ♦ AND 待ち
    指定されたビットが全てセットされるのを待ちます。
  - ♦ OR 待ち
    指定されたビットの内いずれか 1 ビットがセットされるのを待ちます。

- イベントフラグを得る（pol_flg,ipol_flg)
  イベントフラグがあるパターンになっているか否かを調べます。このサービスコールではタスクは待ち状態に移行しません。

- イベントフラグの状態を得る（ref_flg,iref_flg)
  対象イベントフラグのビットパターンや待ちタスクの有無を参照します。

図 3.31に wai_flg と set_flg サービスコールを使用したイベントフラグによるタスクの実行制御の例を示します。

イベントフラグは複数のタスクを一度に起床できるという特徴があります。

図 3.31では、タスク A からタスク F までの 6 個のタスクがつながっています。

そして、set_flg サービスコールによって、フラグパターンを 0x0F にすると、待ち条件にあっているタスクがキューの前から順にはずされていきます。この図で待ち条件を満たすタスクはタスク A、タスク C、タスク E です。このうち、タスク A、タスク C、タスク E がキューからはずされます。したがって、タスク F はキューからはずされません。

もし、本イベントフラグが A_CLR 属性であれば、タスク A の待ちが解除された時点でイベントフラグのビットパターンは 0 となり、タスク C、タスク E はキューからはずされることはありません。

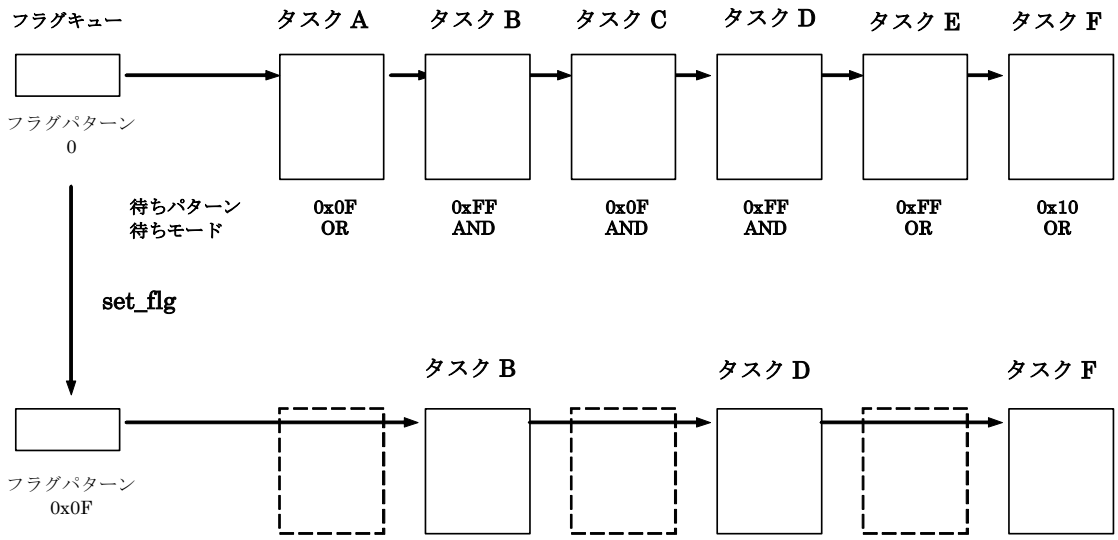


図 3.31 イベントフラグによるタスクの実行制御

3.5.7 同期・通信機能 (データキュー)

データキューとはタスク間でデータの通信をおこなう機構です。例えば、図 3.32においてタスクＡがデータをデータキューに送信しタスクＢがそのデータをデータキューから受信することができます。

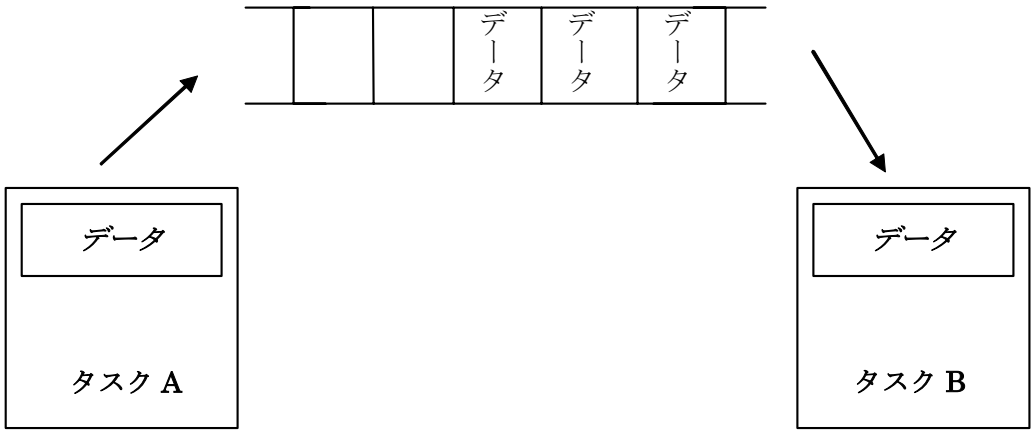


図 3.32 データキュー

このデータキューに送信できるデータ幅は 32 ビットのデータです。

データキューにはデータを蓄積する機能があります。蓄積されたデータは FIFO[31]でデータが取り出されます。ただし、データキューに蓄積できるデータの数には制限があります。データキューがデータで一杯になった状態で、データを送信した場合は、サービスコール発行タスクはデータ送信待ち状態に移行します。

MR308 カーネルが提供するデータキューのサービスコールには次のものがあります。

- データを送信します (snd_dtq, tsnd_dtq)
  データを送信します。すなわち、データをデータキューに送信します。データキューがデータで一杯の場合は、データ送信待ち状態に移行します。

- データを送信します (psnd_dtq, ipsnd_dtq)
  データを送信します。すなわち、データをデータキューに送信します。データキューがデータで一杯の場合は、データ送信待ち状態に移行せず、エラーコードを返します。

- データを受信します (rcv_dtq, trcv_dtq)
  データを受信します。すなわち、データキューからデータを受信します。このときデータキューにデータがなければ、データ受信待ち状態になります。

- データを受信します (prcv_dtq,iprcv_dtq)
  データを受信します。すなわち、データキューからデータを受信します。データキューにデータがない場合は、データ受信待ち状態に移行せず、エラーコードを返します。

- データキューの状態を参照します (ref_dtq,iref_dtq)
  対象データキューにデータが入るのを待っているタスクの有無やデータキューに入っているデータ数を参照します。

[31] ファーストインファーストアウト

3.5.8 同期・通信機能 (メールボックス)

メールボックスとはタスク間でデータの通信をおこなう機構です。例えば、図 3.33においてタスクＡがメッセージをメールボックスに投函しタスクＢがそのメッセージをメールボックスから取り出すことができます。メールボックスを用いた通信は、メッセージの先頭アドレスの受け渡しによって実現されているため、メッセージサイズに依存せずに高速に行われます。

カーネルは、メッセージキューをリンクリストで管理します。アプリケーション側でリンクリストに用いるためのヘッダ領域を用意しなければいけません。これをメッセージヘッダと呼びます。メッセージヘッダと実際にアプリケーションが使用するメッセージを格納する領域をメッセージパケットと呼びます。カーネルはメッセージヘッダの内容を書き換えて管理しています。アプリケーションからはメッセージヘッダを書き換えることはできません。メッセージキューの状態を図 3.34に示します。 メッセージヘッダのデータ型は以下の通り定義しています。

T_MSG: **メールボックスのメッセージヘッダ**
T_MSG_PRI: **メールボックスの優先度付きメッセージヘッダ**

メッセージキューに入れることのできるメッセージのサイズは、アプリケーション側でヘッダ領域を確保するため、制限はありません。また、送信するためにタスクが待ち状態になることはありません。

メッセージに優先度を設定し、優先度の高いメッセージから受信することができます。この場合メールボックス属性に TA_MPRI を付加します。メッセージを FIFO 順に受信する場合はメールボックス属性に TA_MFIFO を付加します。[32]さらに、メッセージ待ち状態のタスクがメッセージを受信する際、優先度の高いタスクからメッセージを受信することもできます。この場合、この場合メールボックス属性に TA_TPRI を付加します。タスクが FIFO 順にメッセージを受信する場合はメールボックス属性に TA_TFIFO を付加します。[33]

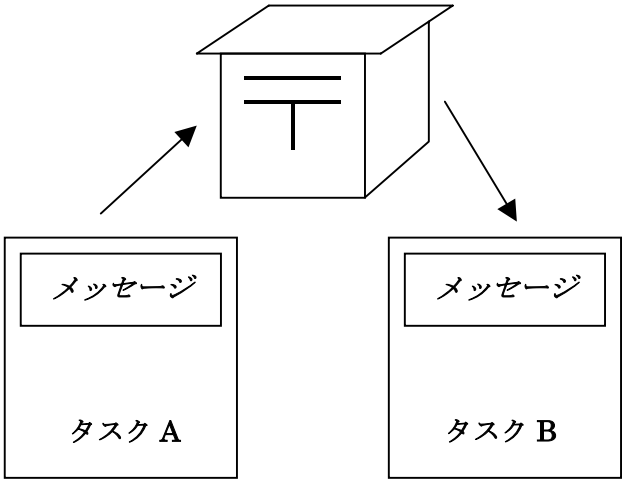


図 3.33 メールボックス

[32]コンフィギュレーションファイルのメールボックス定義”message_queue”項目において TA_MPRI,TA_MFIFO 属性を付加します。
[33] コンフィギュレーションファイルのメールボックス定義”wait_queue”項目において TA_TPRI,TA_TFIFO 属性を付加します。

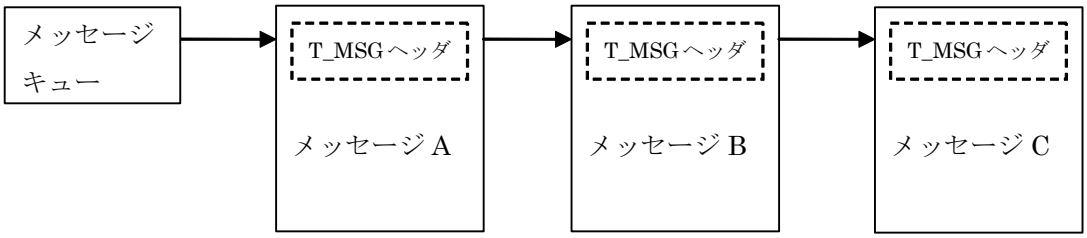


図 3.34 メッセージキュー

MR308 カーネルが提供するメールボックスのサービスコールには次のものがあります。

- メッセージを送信します（snd_mbx, isnd_mbx）
  メッセージを送信します。すなわち、メッセージをメールボックスに投函します。
- メッセージを受信します（rcv_mbx, trcv_mbx）
  メッセージを受信します。すなわち、メッセージをメールボックスから取り出します。このときメッセージがメールボックスに投函されていなければ、投函されるまで待ち状態になります。
- メッセージを受信します（prcv_mbx,iprcv_mbx）
  メッセージを受信します。すなわち、メッセージをメールボックスから取り出します。このときメッセージがメールボックスに投函されていなければ、待ち状態にならずにエラーコードを返します。
- メールボックスの状態を参照します（ref_mbx,iref_mbx）
  対象メールボックスにメッセージが入るのを待っているタスクの有無やメールボックスに入っている先頭のメッセージを参照します。

3.5.9 メモリプール管理機能

メモリプール管理機能はシステムのメモリ空間（RAM 空間）を動的に管理する機能を提供します。
特定のメモリ領域（メモリプール)を管理し、そのメモリプールからタスクあるいはハンドラの必要とするメモリブロックを動的に確保し、また不用になったメモリブロックをメモリプールに解放します。
MR308 では、固定長と可変長の 2 つのメモリプール管理機能をサポートしています。

固定長メモリプール管理機能

メモリプールから獲得できるメモリブロックサイズが決まっていることを固定長といいます。
獲得するメモリブロックサイズは、コンフィギュレーションファイルで指定します。
MR308 カーネルが提供する固定長メモリプール管理サービスコールには次のものがあります。

- メモリブロックを獲得する（get_mpf、tget_mpf)
  指定された ID の固定長メモリプールからメモリブロックを獲得します。空きメモリブロックが、指定された固定長メモリプールにない場合、このサービスコールを発行したタスクは待ち状態に移行し、待ち行列につながれます。

- メモリブロックを獲得する（pget_mpf,ipget_mpf)
  指定された ID の固定長メモリプールからメモリブロックを獲得します。get_mpf、tget_mpf と異なるのは、空きメモリブロックがメモリプールにない場合は、待ち状態に移行せず、エラーコードを返します。

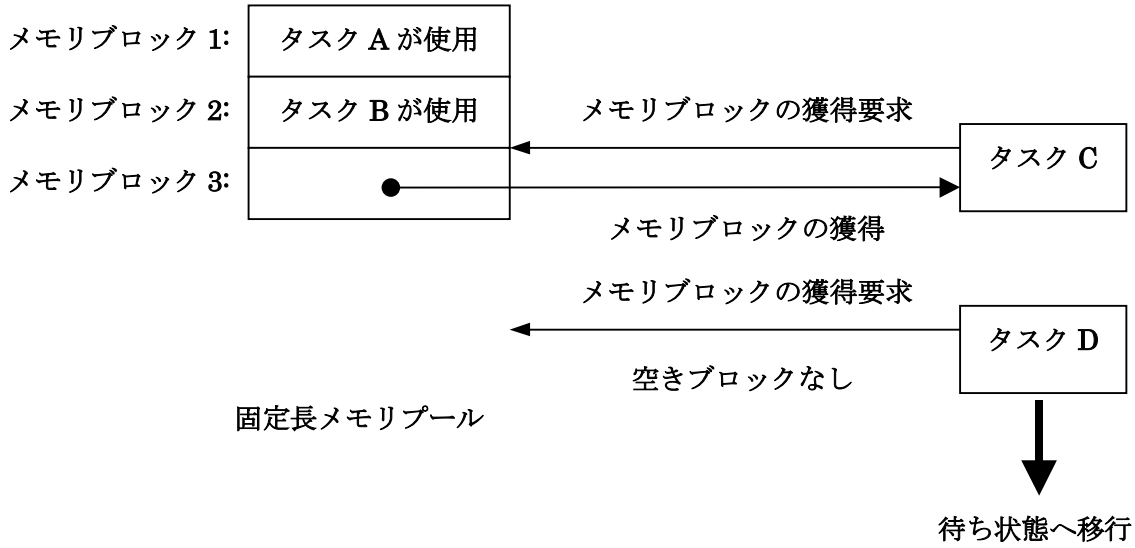


図 3.35 固定長メモリプールの獲得

- メモリブロックを解放する（rel_mpf,irel_mpf)
  獲得しているメモリブロックを解放します。指定された固定長メモリプールに対する待ち状態のタスクがある場合には、待ち行列の先頭につながれたタスクに解放したメモリブロックを割り当てます。この時のタスクの状態は、待ち状態から実行可能(READY)状態に移行します。また、待ち状態のタスクがない場合は、メモリプールにメモリブロックを返却します。

- メモリプールの状態を参照する（ref_mpf,iref_mpf)
  対象メモリプールの空きブロック数やブロックサイズを参照します。

可変長メモリプール管理機能

メモリプールから獲得できるメモリブロックサイズが任意に指定可能なことを可変長といいます。

MR308 では、メモリを 4 種類の固定長ブロックサイズで管理しています。

4 種類の各々のサイズは、ユーザーが獲得するメモリブロックの最大サイズから MR308 が計算します。メモリブロックの最大サイズは、コンフィギュレーションファイルで指定します。

例

```
variable_memorypool[]{
    max_memsize = 400; <----- 最大獲得サイズ
    heap_size = 5000;
};
```

上記のように、可変長メモリプール定義を行った場合、4 種類の固定長ブロックサイズは、 max_memsize 定義値から 56,112,224,448 となります。

ユーザーが要求したメモリは、指定サイズをもとに MR308 が計算を行い 4 種類の固定長メモリブロックサイズの中から最適なサイズを選択し、メモリを割り当てます。この 4 種類以外のサイズのメモリブロックを割り当てることはありません。

MR308 カーネルが提供する可変長メモリプール管理サービスコールには次のものがあります。

- メモリブロックを獲得する（pget_mpl）

  ユーザーが指定したブロックサイズは、 4 種類のブロックサイズのうちから最適なブロックサイズに丸めて、丸めたサイズ分のメモリをメモリプールから獲得します。

  4 種類のブロックサイズ(下記 a,b,c,d）は下記の計算式から算出されます。

  ```
  a = (((max_memsize+(X-1))/(X×8))+1) ×X
  b = a × 2
  c = a × 4
  d = a × 8
  ```

  max_memsize：コンフィギュレーションファイルで指定した値

  X：ブロック管理用データサイズ（8 バイト）

  例えば、ユーザーが 200 バイトのメモリを要求した場合 224 バイトに丸めて、224 バイト分のメモリを獲得します。

  メモリが獲得できた場合、獲得したメモリの先頭アドレスとエラーコード E_OK を返します。 獲得できなかった場合には、エラーコード E_TMOUT を返します。

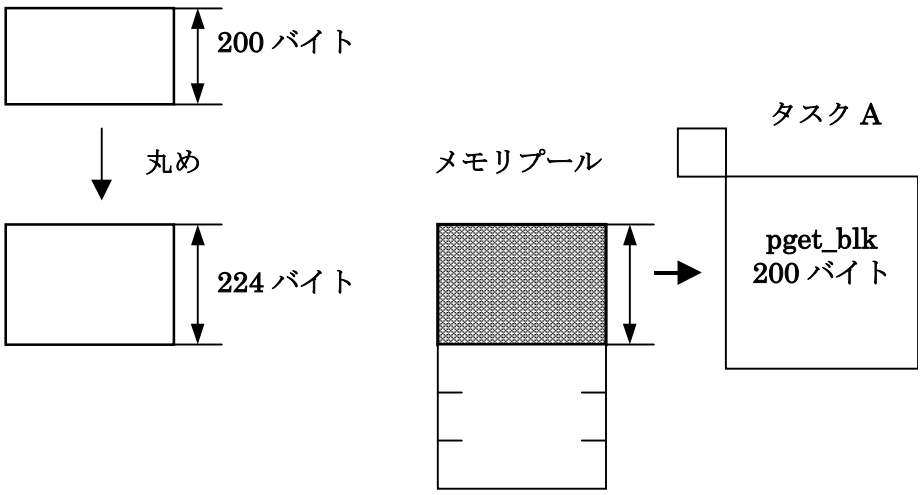


図 3.36 pget_blk 処理

- メモリブロックを解放する（rel_mpl）
  pget_mpl で獲得したメモリブロックを解放します。

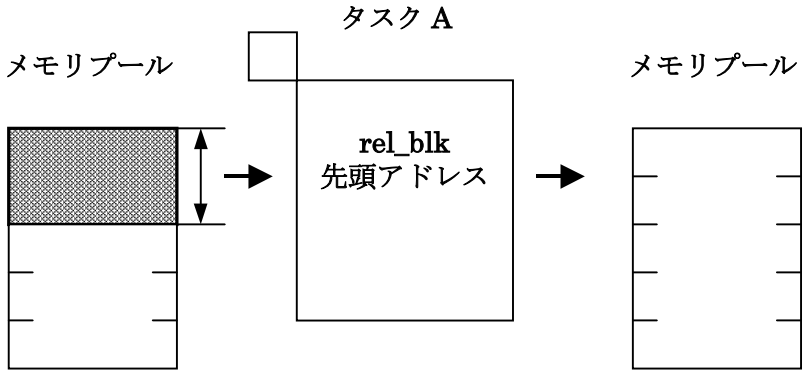


図 3.37 rel_blk 処理

- メモリプールの状態を参照する（ref_mpl,iref_mpl）
  メモリプールの空き領域の合計サイズやすぐに獲得できる最大の空き領域のサイズを参照します。

3.5.10 時間管理機能

時間管理機能はシステムの時刻を管理し、時刻の読みだし[34]、時刻の設定[35]機能、タイムアウトの処理や特定時刻に起動するアラームハンドラや定期的に起動する周期ハンドラの機能を提供します。

MR308 カーネルはシステムクロックとしてタイマを一つ必要とします。MR308 カーネルが提供する時間管理サービスコールには次のものがあります。なお、システムクロックは必須機能ではありません。したがって下記のサービスコールおよび時間管理機能を使用しなければ、タイマを MR308 用に占有する必要がありません。

- 待ち状態にタイムアウト値を指定すれば、一定時間待ち状態に移行します
  タスクを待ち状態に移行するサービスコール[36]にタイムアウトを指定することができます。サービスコール名は、tslp_tsk、twai_flg、twai_sem、tsnd_dtq、trcv_dtq、trcv_mbx、tget_mpf、vtsnd_dtq、vtrcv_dtq です。タイムアウトの指定時間が経過するまでに待ち解除条件が満たされない場合、エラーコード E_TMOUT を返し、待ち状態が解除されます。待ち解除条件が満たされた場合は、エラーコードは E_OK を返します。
  タイムアウトの時間単位は、ms です。

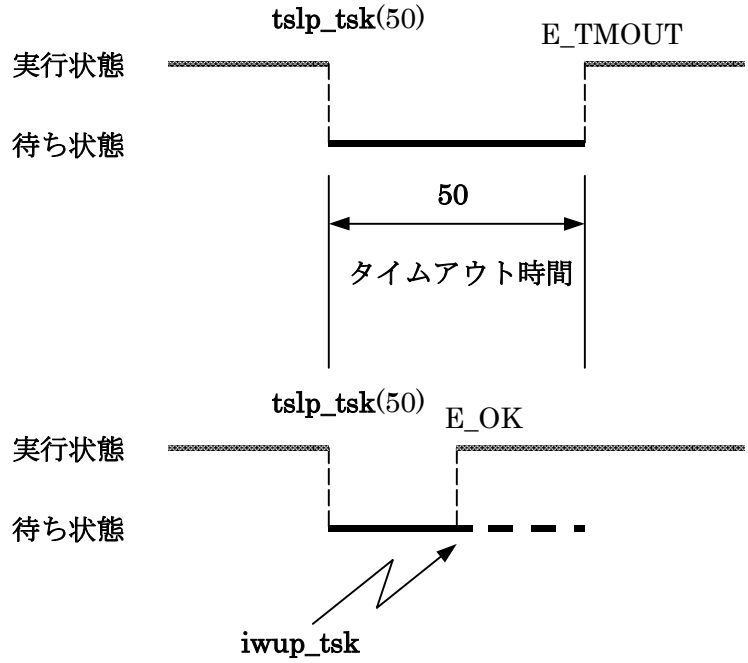


図 3.38 タイムアウト処理

MR308 では、μITRON 仕様の規定通り、指定されたタイムアウト値分以上の時間が経過してからタイムアウト処理を行うことを保証します。具体的には以下のタイミングでタイムアウト処理を行います。

1. タイムアウト値が 0 の場合(dly_tsk の場合のみ)
   サービスコール発行後の最初のタイムティックでタイムアウトします。
2. タイムアウト値が、タイムティック間隔の倍数である場合
   (タイムアウト値/タイムティック間隔)+1 回目のタイムティックでタイムアウトします。例えば、タイムティック間隔が 10ms でタイムアウト値に 40ms を指定した場合、5 回目のタイムティックでタイムアウトします。また、タイムティック間隔が 5ms でタイムアウト値に 15ms を指定した場合、4 回目のタイムティックでタイムアウトします。
3. タイムアウト値が、タイムティック間隔の倍数でない場合
   (タイムアウト値/タイムティック間隔)+2 回目のタイムティックでタイムアウトします。例えば、

[34] get_tim サービスコール
[35] set_tim サービスコール
[36] 強制待ち状態を除きます。

タイムティック間隔が 10ms でタイムアウト値に 35ms を指定した場合、5 回目のタイムティックでタイムアウトします。

- システム時刻を設定する（set_tim）
- システム時刻の値を読みだす（get_tim）
  システム時刻はリセット時からの経過時間を 48 ビットのデータで表します。時間の単位は ms です。

3.5.11 周期ハンドラ機能

周期ハンドラは、指定した起動位相経過後、起動周期ごとに起動されるタイムイベントハンドラです。

周期ハンドラの起動には、起動位相を保存する方法と起動位相を保存しない方法があります。起動位相を保存する場合は、周期ハンドラの生成時点を基準に周期ハンドラを起動します。起動位相を保存しない場合は、周期ハンドラの動作開始時点を基準に周期ハンドラを起動します。図 3.39、図 3.40に周期ハンドラの動作例を示します。

タイムティック間隔より、起動周期が短い場合、タイムティック供給(isig_tim 相当の処理)毎に、1 回だけ周期ハンドラを起動します。例えば、タイムティック間隔が 10ms、起動周期が 3ms で、タイムティック供給時に周期ハンドラ動作が開始された場合、タイムティックごとに 1 回だけ周期ハンドラが起動されることになります。

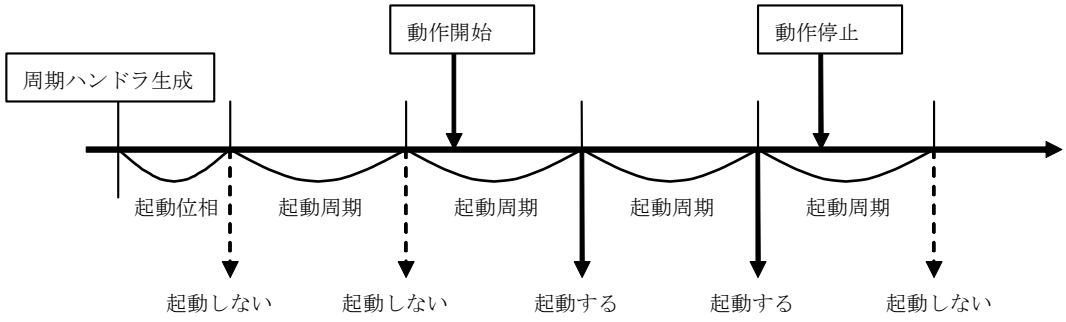


図 3.39 起動位相を保存する場合の動作

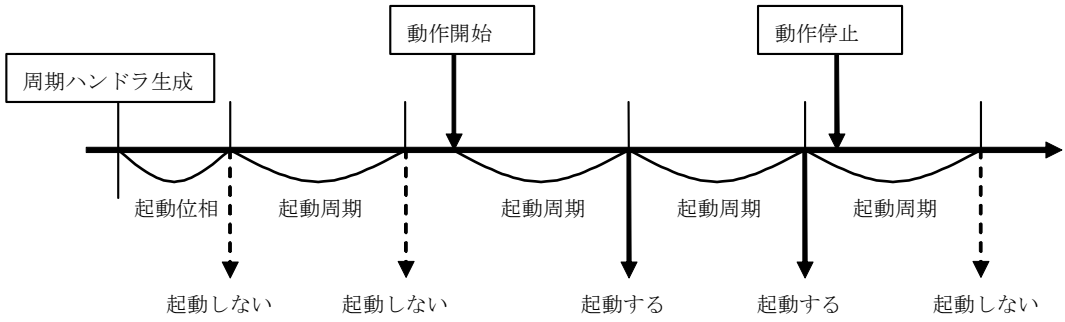


図 3.40 起動位相を保存しない場合の動作

- 周期ハンドラの動作を開始する(sta_cyc,ista_cyc)
  指定された ID の周期ハンドラの動作を開始します。
- 周期ハンドラの動作を停止する(stp_cyc,istp_cyc)
  指定された ID の周期ハンドラの動作を停止します。
- 周期ハンドラの状態を参照する (ref_cyc,iref_cyc)
  周期ハンドラの状態を参照します。対象周期ハンドラの動作状態と次の起動までの残り時間を調べます。

3.5.12 アラームハンドラ機能

アラームハンドラは、指定した時刻になると 1 度だけ起動されるタイムイベントハンドラです。
アラームハンドラを用いることにより、時刻に依存した処理を行うことができます。時刻の指定は、相対時間です。図 3.41 動作例をに示します。

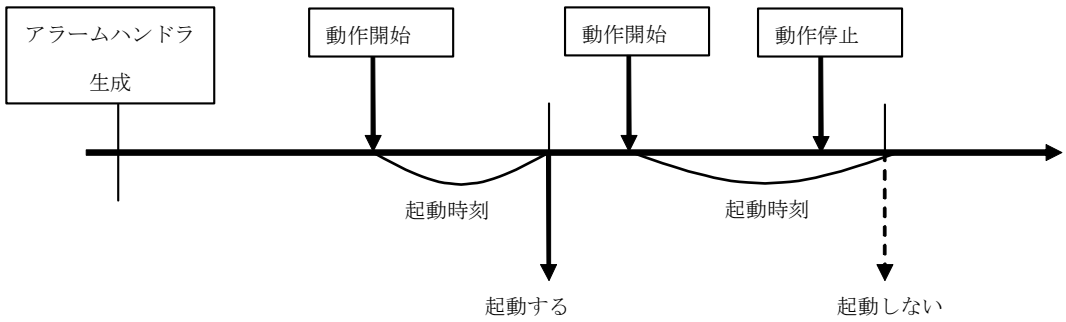


図 3.41 アラームハンドラの動作

- アラームハンドラの動作を開始する(sta_cyc,ista_cyc)
  指定された ID のアラームハンドラの動作を開始します。
- アラームハンドラの動作を停止する(stp_cyc,istp_cyc)
  指定された ID のアラームハンドラの動作を停止します。
- アラームハンドラの状態を参照する (ref_cyc,iref_cyc)
  アラームハンドラの状態を参照します。対象アラームハンドラの動作状態と起動までの残り時間を調べます。

3.5.13 システム状態管理機能

タスクの実行待ち行列を回転する（rot_rdq , irot_rdq）

本サービスコールにより TSS(タイムシェアリングシステム) を実現することができます。すなわち、一定周期でレディキューを回転すれば、TSS で必要なラウンドロビンスケジューリングを実現することができます。 (図 3.42参照)

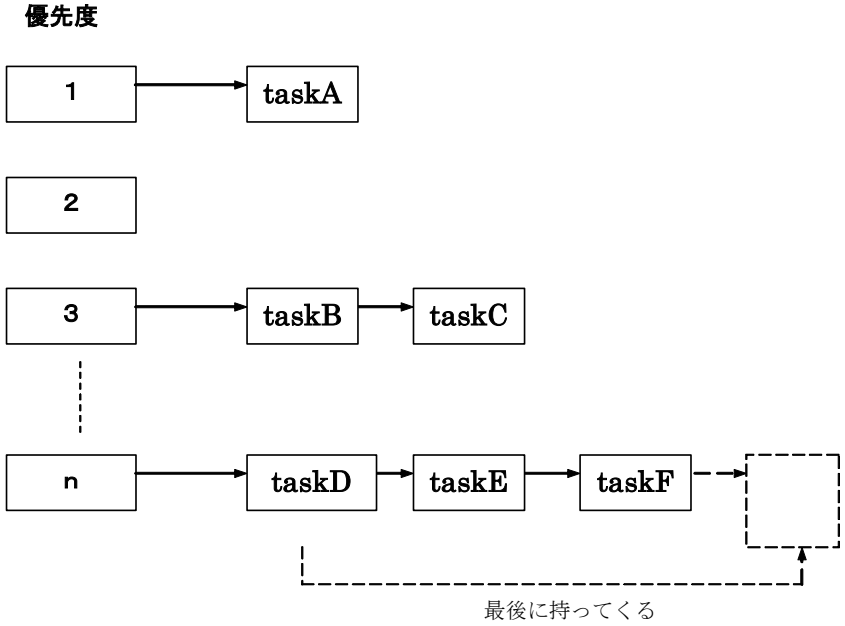


図 3.42 rot_rdq サービスコールによるレディキューの操作

- 自タスクの ID を得る（get_tid,iget_tid）
  自タスクの ID 番号を得ます。ハンドラから発行した場合は、ID 番号の替わりに TSK_NONE(=0=) が得られます。

- CPU ロック状態に移行する（loc_cpu,iloc_cpu）
  CPU ロック状態に移行します。

- CPU ロック状態を解除する（unl_cpu,iunl_cpu）
  CPU ロック状態を解除します。

- ディスパッチ禁止状態に移行する（dis_dsp）
  ディスパッチ禁止状態に移行します。

- ディスパッチ禁止状態を解除する（ena_dsp）
  ディスパッチ禁止状態を解除します。

- コンテキスト状態を得ます（sns_ctx）
  コンテキスト状態を得ます。

- CPU ロック状態を得ます（sns_loc）
  CPU ロック状態を得ます。

- ディスパッチ禁止状態を得ます（sns_dsp）
  ディスパッチ禁止状態を得ます。

- ディスパッチ保留状態を得ます（sns_dpn）
  ディスパッチ保留状態を得ます。

3.5.14 割り込み管理機能

割り込み管理機能は外部割り込みの発生に対して、実時間で処理をおこなう機能を提供します。
MR308 カーネルが提供する割り込み管理サービスコールには次のものがあります。

- 割り込みハンドラから復帰します (ret_int)
  ret_int サービスコールは、割り込みハンドラから復帰するとき、必要ならばスケジューラを起動し、タスク切り替えをおこないます。
  本機能は、C 言語を用いた場合[37]、ハンドラ関数の終了時に自動で呼び出されます。従って、この場合、本サービスコールを呼び出す必要はありません。

図 3.43に割り込み処理の流れをしめします。なお、タスク選択からレジスタ復帰までの処理をスケジューラと呼びます。

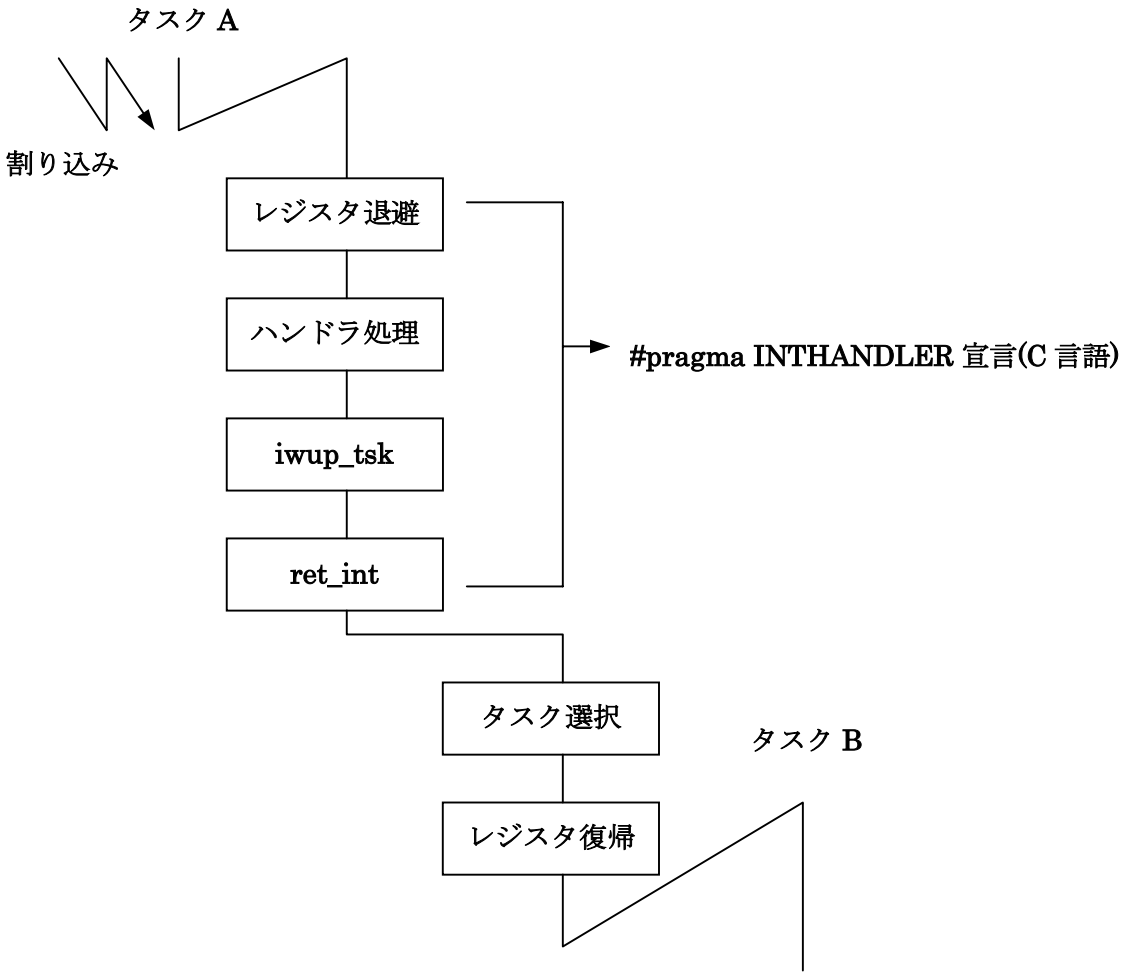


図 3.43 割り込み処理の流れ

[37] 割り込みハンドラを、#pragrma INTHANDLER で指定した場合

3.5.15 システム構成管理機能

MR308 のバージョン情報を参照する機能です。

- MR308 のバージョンを得る(ref_ver,iref_ver)
  MR308 のバージョンを ref_ver サービスコールにより得ることができます。このバージョンはμITRON 仕様で標準化された形式で得ることができます。

3.5.16 拡張機能(short データキュー)

short データキューは、μITRON 4.0 仕様の仕様外の機能です。データキュー機能は、データを 32bit として扱いますが、short データキューは、データを 16bit データとして扱います。両者の動作はデータサイズの差異以外は違いがありません。

- データを送信します（vsnd_dtq, vtsnd_dtq）
  データを送信します。すなわち、データを short データキューに送信します。short データキューがデータで一杯の場合は、データ送信待ち状態に移行します。

- データを送信します（vpsnd_dtq, vipsnd_dtq）
  データを送信します。すなわち、データを short データキューに送信します。short データキューがデータで一杯の場合は、データ送信待ち状態に移行せず、エラーコードを返します。

- データを受信します（vrcv_dtq, vtrcv_dtq）
  データを受信します。すなわち、すなわち、short データキューからデータを受信します。このとき short データキューにデータがなければ、データが送信されるまで待ち状態になります。

- データを受信します（vprcv_dtq,viprcv_dtq）
  データを受信します。すなわち、short データキューからデータを受信します。short データキューにデータがない場合は、データ受信待ち状態に移行せず、エラーコードを返します。

- データキューの状態を参照します（vref_dtq,viref_dtq）
  対象 short データキューにデータが入るのを待っているタスクの有無や short データキューに入っているデータ数を参照します。

3.5.17 拡張機能(リセット機能)

リセット機能は、μITRON 4.0 仕様の仕様外の機能です。メールボックス、データキュー、メモリプールなどを初期状態にします。

- データキューを初期化する(vrst_dtq)
  データキューを初期化します。送信待ちのタスクがある場合は、待ち状態を解除し、エラーコード EV_RST を返します。

- メールボックスを初期化する(vrst_mbx)
  メールボックスを初期化します。

- 固定長メモリプールを初期化する(vrst_mpf)
  固定長メモリプールを初期化します。待ち状態のタスクがある場合は、待ち状態を解除し、エラーコード EV_RST を返します。

- 可変長メモリプールを初期化する(vrst_mpl)
  可変長メモリプールを初期化します。

- short データキューを初期化する(vrst_vdtq)
  short データキューを初期化します。送信待ちのタスクがある場合は、待ち状態を解除し、エラーコード EV_RST を返します。

3.5.18 タスクコンテキスト、非タスクコンテキストから発行できるサービスコール一覧

サービスコールにはタスクから発行できるものと非タスクコンテキストから発行できるもの、その両方から発行できるものがあります。表 3-4にその一覧を示します。

表 3-4 タスクコンテキスト、非タスクコンテキストから発行できるサービスコール一覧

サービスコール	タスクコンテキスト	非タスクコンテキスト
act_tsk	○	×
iact_tsk	×	○
can_act	○	×
ican_act	×	○
sta_tsk	○	×
ista_tsk	×	○
ext_tsk	○	×
ter_tsk	○	×
chg_pri	○	×
ichg_pri	×	○
get_pri	○	×
iget_pri	×	○
ref_tsk	○	×
iref_tsk	×	○
ref_tst	○	×
iref_tst	×	○
slp_tsk	○	×
tslp_tsk	○	×
wup_tsk	○	×
iwup_tsk	×	○
can_wup	○	×
ican_wup	×	○
rel_wai	○	×
irel_wai	×	○
sus_tsk	○	×
isus_tsk	×	○
rsm_tsk	○	×
irsm_tsk	×	○
frsm_tsk	○	×
ifrsm_tsk	×	○
dly_tsk	○	×

サービスコール	タスクコンテキスト	非タスクコンテキスト
sig_sem	○	×
isig_sem	×	○
wai_sem	○	×
twai_sem	○	×
pol_sem	○	×
ipol_sem	×	○
ref_sem	○	×
iref_sem	×	○
set_flg	○	×
iset_flg	×	○
clr_flg	○	×
iclr_flg	×	○
wai_flg	○	×
twai_flg	○	×
pol_flg	○	×
ipol_flg	×	○
ref_flg	○	×
iref_flg	×	○
snd_dtq	○	×
tsnd_dtq	○	×
psnd_dtq	○	×
ipsnd_dtq	×	○
fsnd_dtq	○	×
ifsnd_dtq	×	○
rcv_dtq	○	×
trcv_dtq	○	×
prcv_dtq	○	×
iprcv_dtq	×	○
ref_dtq	○	×
iref_dtq	×	○
snd_mbx	○	×
isnd_mbx	×	○
rcv_mbx	○	×
trcv_mbx	○	×
prcv_mbx	○	×
iprcv_mbx	×	○
ref_mbx	○	×
iref_mbx	×	○

サービスコール	タスクコンテキスト	非タスクコンテキスト
get_mpf	○	×
tget_mpf	○	×
pget_mpf	○	×
rel_mpf	○	×
irel_mpf	×	○
ref_mpf	○	×
iref_mpf	×	○
ipget_mpf	×	○
pget_mpl	○	×
rel_mpl	○	×
ref_mpl	○	×
iref_mpl	×	○
set_tim	○	×
iset_tim	×	○
get_tim	○	×
iget_tim	×	○
sta_cyc	○	×
ista_cyc	×	○
stp_cyc	○	×
istp_cyc	×	○
ref_cyc	○	×
iref_cyc	×	○
sta_alm	○	×
ista_alm	×	○
stp_alm	○	×
istp_alm	×	○
ref_alm	○	×
iref_alm	×	○
rot_rdq	○	×
irot_rdq	×	○
get_tid	○	×
iget_tid	×	○
loc_cpu	○	×
iloc_cpu	×	○
unl_cpu	○	×
iunl_cpu	×	○
dis_dsp	○	×
ena_dsp	○	×
sns_ctx	○	○
sns_loc	○	○
sns_dsp	○	○
sns_dpn	○	○

サービスコール	タスクコンテキスト	非タスクコンテキスト
ref_ver	○	×
iref_ver	×	○
vsnd_dtq	○	×
vtsnd_dtq	○	×
vpsnd_dtq	○	×
vipsnd_dtq	×	○
vfsnd_dtq	○	×
vifsnd_dtq	×	○
vrcv_dtq	○	×
vtrcv_dtq	○	×
vprcv_dtq	○	×
viprcv_dtq	×	○
vref_dtq	○	×
viref_dtq	×	○
vrst_mpf	○	×
vrst_mpl	○	×
vrst_mbx	○	×
vrst_dtq	○	×
vrst_vdtq	○	×

第 4 章　　アプリケーション作成手順概要

4.1 概要

MR308 のアプリケーションプログラムは一般的に以下に示す手順で開発します。

1. プロジェクトの生成

HEW[38]を使用する場合は、HEW 上で MR308 を使用したプロジェクトを新規に作成します。

2. アプリケーションプログラムのコーディング

C 言語もしくはアセンブリ言語を用いてアプリケーションプログラムをコーディングします。

必要があればサンプルのスタートアッププログラム(crt0mr.a30)、セクション定義ファイル(c_sec.inc もしくは asm_sec.inc)を修正してください。

3. コンフィギュレーションファイル作成

タスクのエントリーアドレスやスタックサイズなどを定義したコンフィギュレーションファイルをエディタなどで作成します。

GUI コンフィギュレータを用いてコンフィギュレーションファイルを作成することも出来ます。

4. コンフィギュレータ実行

コンフィギュレーションファイルからシステムデータ定義ファイル（sys_rom.inc、sys_ram.inc)、インクルードファイル（mr308.inc、kernel_id.h)およびシステム生成手順記述ファイル（makefile）を作成します。

5. システム生成

make[39]コマンドもしくは HEW 上でビルドを実行してシステムを生成します。

6. ROM 書き込み

作成された ROM 書き込み形式ファイルにより、ROM に書き込みます。もしくはデバッガに読み込ませてデバッグを行います。

図 4.1にシステム生成の詳細フローを示します。

[38] High-performance Embedded Workshop の略称です。

[39] Make コマンドは、UNIX 標準もしくは準拠のコマンドのみ使用可能です。

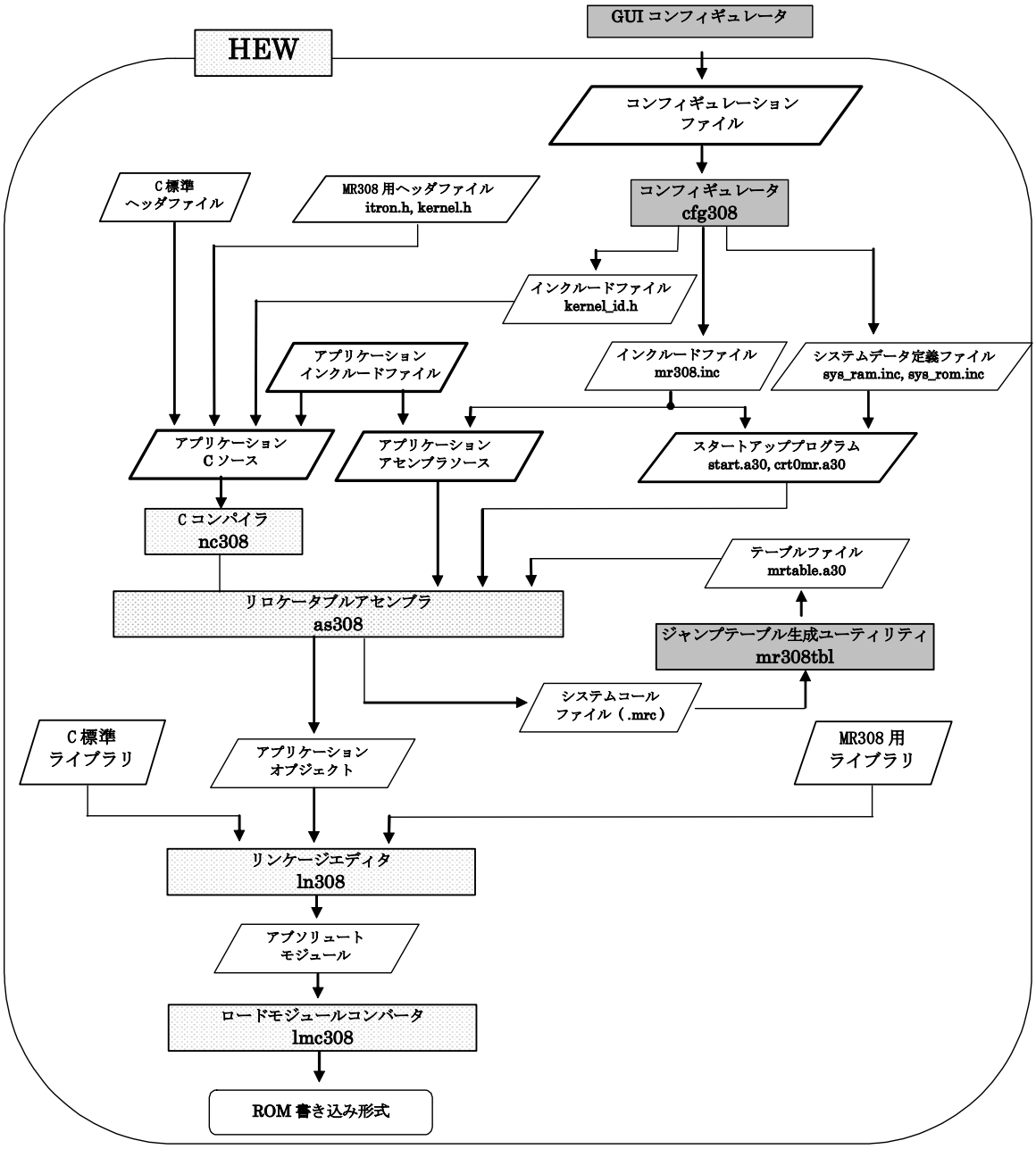


図 4.1　MR308 システム生成詳細フロー

4.2 開発手順例

この節では MR308 のアプリケーション例をもとに開発手順の概要について説明します。

4.2.1 アプリケーションプログラムのコーディング

図 4.2にレーザビームプリンタの動作をシミュレーションするプログラムを示します。このレーザービームプリンタのシミュレーションプログラムを記述したファイルの名前を"lbp.c"とします。このプログラムは以下の３つのタスクと１つの割り込みハンドラから構成されます。

- メインタスク
- イメージ展開タスク
- プリンタエンジンタスク
- セントロニクスインターフェース割り込みハンドラ

このプログラムでは MR308 ライブラリのなかの以下の機能を利用します。

- sta_tsk()
  タスク起動をおこないます。引き数は起動すべきタスクを選択するための ID 番号を与えます。ＩＤ番号はコンフィギュレータの生成するファイル"kernel_id.h"をインクルードすることにより名前（文字列)でタスクを指定することができます。[40]
- wai_flg()
  イベントフラグが立つまで待ちます。この例ではセントロニクスインターフェースから１ページ分データがバッファに溜まるのを待つために使用しています。
- wup_tsk()
  指定タスクを待ち状態から起床します。プリンタエンジンタスクを起動するために使用しています。
- slp_tsk()
  タスクを実行状態から待ち状態にします。この例ではプリンタエンジンタスクをイメージ展開待ちにするため使用しています。
- iset_flg()
  イベントフラグを立てます。この例では、１ページ分のデータ入力完了をイメージ展開タスクに知らせるために使用しています。

[40] すなわち、コンフィギュレータがコンフィギュレーションファイルに記述されている情報をもとに ID 番号を名前（文字列）に置き換えます。

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"


void main() /* main task */
{
    printf("LBP シミュレーション開始¥n");
    sta_tsk(ID_idle,1);     /* アイドルタスク起動 */
    sta_tsk(ID_image,1);    /* イメージ展開タスク起動 */
    sta_tsk(ID_printer,1);  /* プリンタエンジンタスク起動 */
}
void image() /* イメージ展開タスク */
{
    while(1){
            wai_flg(ID_pagein,waiptn,TWF_ANDW, &flgptn);/* 1ページ入力待ち */

            printf("ビットマップ展開処理¥n");
            wup_tsk(ID_printer);    /* プリンタエンジンタスク起床 */
    }
}
void printer() /* プリンタエンジンタスク */
{
    while(1){
            slp_tsk();
            printf("プリンタエンジン動作¥n");
    }
}
void sent_in() /* セントロニクスインターフェースハンドラ */
{


    /* セントロニクスインターフェースからの入力処理 */
    if ( /* 1ページ入力完了 */ )
            iset_flg(ID_pagein,setptn);
}
```

図 4.2　プログラム例

4.2.2 コンフィギュレーションファイル作成

タスクのエントリーアドレスやスタックサイズなどを定義したコンフィギュレーションファイルを作成します。GUI コンフィギュレータを使用することによってコンフィギュレーションファイルの記述形式を習得しなくとも容易に作成することが出来ます。

図 4.3にレーザービームプリンタシュミレーションプログラムのコンフィギュレーションファイル（ファイル名 "lbp.cfg"）を示します。

```
// System Definition
system{
    stack_size              = 1024;
    priority                = 5;
    system_IPL              = 4;
    tick_nume               = 10;
};
//System Clock Definition
clock{
    mpu_clock               = 20MHz;
    timer                   = A0;
    IPL                     = 4;
};
//Task Definition
task[1]{
    name                    = ID_main;
    entry_address           = main();
    stack_size              = 512;
    priority                = 1;
    initial_start           = ON;
};
task[2]{
    name                    = ID_image;
    entry_address           = image();
    stack_size              = 512;
    priority                = 2;
};
task[3]{
    name                    = ID_printer;
    entry_address           = printer();
    stack_size              = 512;
    priority                = 4;
};
task[4]{
    name                    = ID_idle;
    entry_address           = idle();
    stack_size              = 256;
    priority                = 5;
};
//Eventflag Definition
flag[1]{
     name                   = pagein;
};
//Interrupt Vector Definition
interrupt_vector[0x23]{
    os_int                  = YES;
    entry_address           = sent_in();
};
```

図 4.3　コンフィギュレーションファイル例

4.2.3 コンフィギュレータ実行

HEW を使用する場合は、「すべてをビルド」を選択することにより、「4.2.3 コンフィギュレータ実行」「4.2.4 システム生成」の手順を実施することが出来ます。

コンフィギュレータ cfg308 を実行して、コンフィギュレーションファイルからシステムデータ定義ファイル (sys_rom.inc, sys_ram.inc) 、インクルードファイル (mr308.inc, kernel_id.h)
およびシステム生成手順記述ファイル (makefile) を作成します。

```
A> cfg308  -mv  lbp.cfg

MR308 system configurator V.4.00.06
Copyright 2003,2005 RENESAS TECHNOLOGY CORPORATION
AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED.
MR308 version ==> V.4.00 Release 01

A>
```

図 4.4 コンフィギュレータ実行

4.2.4 システム生成

make コマンド[41]を実行してシステムを生成します。

```
A> make -f  makefile
as308 -F -Dtest=1 crt0mr.a30
nc308 -c task.c
ln308 @ln308.sub

A>
```

図 4.5 システム生成

4.2.5 ROM 書き込み

ロードモジュールコンバータ lmc308 により、アブソリュートモジュールファイルを ROM 書き込み形式に変換し、ROM に書き込みます。もしくは、デバッガに読み込ませてデバッグを行います。

[41] make コマンドは MS-DOS 標準のものと、UNIX 標準もしくは準拠のものとがあります。MR308 では、UNIX 標準もしくは UNIX 準拠の make コマンドのみ使用できます。MS-DOS を使用する場合は、UNIX 互換の make コマンド（例えば、マイクロソフト社製Cコンパイラ付属の nmake コマンド）を使用してください。UNIX 互換の make コマンド対応状況については、リリースノートを参照下さい。本章では、UNIX 互換の nmake コマンドを実行する場合を例として説明します。

第 5 章　　アプリケーション作成手順詳細

5.1 C 言語によるコーディング方法

本節では、C 言語を用いてアプリケーションプログラムを記述する方法について述べます。

5.1.1 タスクの記述方法

C 言語を用いてタスクを記述する場合、以下の項目に注意してください。

1. **タスクは関数として記述します。**
   そのタスクを MR308 に登録するにはコンフィギュレーションファイルに関数名を記述します。例えば関数名 "task()" をタスク ID 番号３で登録するには以下のようにおこないます。

```
task[3]{
        name            = ID_task;

        entry_address   = task();

        stack_size      = 100;

        priority                = 3;

};
```

2. **ファイル先頭で必ずシステムディレクトリのなかの "itron.h","kernel.h" とカレントディレクトリ内の "kernel_id.h" をインクルードしてください。すなわちファイルの先頭で以下の３行を必ず記述してください。**

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
```

3. **タスク開始関数の戻り値はありません。したがって、void 型で宣言してください。**
4. **スタティック宣言をおこなった関数はタスクとして登録できません。**
5. **タスク開始関数の出口で ext_tsk()を記述する必要はありません。[42]タスク開始関数から呼び出した関数でタスクを終了する場合は、ext_tsk()を記述してください。**
6. **タスクの開始関数を無限ループで記述することも可能です。**

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"

void task(VP_INT stacd)
{

        /* 処理 */

}
```

図 5.1　C 言語で記述したタスクの例

[42] MR308 では、#pragma TASK 宣言を行うことで、自動的に ext_tsk()で終了します。関数の途中で return 文により戻る場合も同様に ext_tsk()で終了処理をおこないます。

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"

void task(VP_INT stacd)
{
        for(;;){
                /* 処理 */

        }
}
```

図 5.2　C 言語で記述した無限ループタスクの例

7. **タスクを指定する場合はコンフィギュレーションファイルのタスク定義の項目”name”に記述した文字列で指定してください[43]。**

```
wup_tsk(ID_main);
```

8. **イベントフラグ、セマフォ、メールボックスを指定する場合は、コンフィギュレーションファイルで定義したそれぞれの名前の文字列で指定してください。**

   例えば、コンフィギュレーションファイルで以下のようにイベントフラグを定義した場合は、

```
flag[1]{
        name    = ID_abc;
};
```

   このイベントフラグを指定するには以下のようにおこなってください。

```
set_flg(ID_abc,(FLGPTN)setptn);
```

9. **周期ハンドラ、アラームハンドラを指定する場合は、コンフィギュレーションファイルのアラームハンドラもしくは周期ハンドラ定義の項目”name”に記述した文字列で指定してください。**

```
sta_cyc(ID_cyc);
```

10. **タスクを ter_tsk() サービスコールなどで終了した後で sta_tsk() サービスコールなどで再起動した場合は、タスク自身は初期状態から開始しますが[44]外部変数、スタティック変数はタスクの開始にともなっての初期化はされません。外部変数、スタティック変数の初期化は MR308 が立ち上がる前に起動されるスタートアッププログラム（crt0mr.a30、start.a30）でのみおこないます。**

11. **MR308 システム起動時に起動されるタスクは、コンフィギュレーションファイルで設定します。**

12. **変数の記憶クラスについて**

    C 言語の変数は MR308 から見て表 5.1に示す扱いになります。

[43] コンフィギュレータがタスクの ID 番号をタスクを指定するための文字列に変換するためのファイル”kernel_id.h”を生成します。すなわち、タスク定義の項目”name”に指定した文字列をそのタスクの ID 番号に変換するための#define 宣言を”kernel_id.h”で行います。周期ハンドラ,アラームハンドラも同様です。

[44] タスクの開始関数から初期優先度でなおかつ起床カウントがクリアされた状態で開始します。

表 5.1　C 言語における変数の扱い

変数の記憶クラス	扱い
グローバル変数	すべてのタスクの共有変数
関数外のスタティック変数	同一ファイル内のタスクの共有変数
オート変数 レジスタ変数 関数内のスタティック変数	タスク固有の変数

5.1.2 カーネル管理(OS 依存)割り込みハンドラの記述方法

C 言語を用いてカーネル管理(OS 依存)割り込みハンドラを記述する場合、以下の点に注意してください。

1. **カーネル管理(OS 依存)割り込みハンドラは関数として記述します。[45]**
2. **割り込みハンドラ開始関数の戻り値および引き数は、必ず void 型で宣言してください。**
3. **ファイル先頭で必ずシステムディレクトリのなかの "itron.h","kernel.h" とカレントディレクトリ内の "kernel_id.h" をインクルードしてください。すなわちファイルの先頭で以下の３行を必ず記述してください。**

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
```

4. **関数の最後に ret_int サービスコールは記述しないで下さい。[46]また、割り込みハンドラ関数から呼び出した関数のなかで割り込みハンドラを終了することはできません。**
5. **スタティック宣言をおこなった関数は割り込みハンドラとしては登録できません。**

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"

void inthand(void)

{
        /* 処理 */

        iwup_tsk(ID_main);
}
```

図 5.3 カーネル管理(OS 依存)割り込みハンドラの例

[45] ハンドラと関数名との対応はコンフィギュレーションファイルにより行います。
[46] カーネル管理(OS 依存)割り込みハンドラを#pragma INTHANDLER で宣言すると、自動的に ret_int サービスコールを生成します。

5.1.3 カーネル管理外(OS 独立)割り込みハンドラの記述方法

C 言語を用いてカーネル管理外(OS 独立)割り込みハンドラを記述する場合、以下の点に注意してください。

1. **割り込みハンドラ開始関数の戻り値および引き数は、必ず void 型で宣言してください。**
2. **カーネル管理外(OS 独立)割り込みハンドラからは、サービスコールは発行できません。**
   (注)サービスコールを発行した場合は不正動作をするので十分注意して下さい。
3. **スタティック宣言をおこなった関数は割り込みハンドラとしては登録できません。**
4. **カーネル管理外(OS 独立)割り込みハンドラの中で多重割り込みを許可する場合は、必ず、カーネル管理外(OS 独立)割り込みハンドラの割り込み優先レベルを、他のカーネル管理(OS 依存)割り込みハンドラの割り込み優先レベルより高くしてください。[47]**

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"

void inthand(void)
{
        /* 処理 */
}
```

図 5.4 カーネル管理外(OS 独立)割り込みハンドラの例

[47] カーネル管理外(OS 独立)割り込みハンドラの割り込みレベル優先レベルを、カーネル管理(OS 依存)割り込みハンドラの割り込み優先レベルより低くしたい場合は、カーネル管理外(OS 独立)割り込みハンドラをカーネル管理(OS 依存)割り込みハンドラの記述に変更してください。

5.1.4 周期ハンドラ、アラームハンドラの記述方法

C 言語を用いて周期ハンドラおよびアラームハンドラを記述する場合、以下の点に注意してください。

1. **周期ハンドラおよびアラームハンドラは関数として記述します。**[48]
2. **関数の引数を VP_INT 型、戻り値は void 型で宣言してください。**
3. **ファイル先頭で必ずシステムディレクトリのなかの "itron.h","kernel.h" とカレントディレクトリ内の "kernel_id.h" をインクルードしてください。すなわちファイルの先頭で以下の３行を必ず記述してください。スタティック宣言をおこなった関数は周期ハンドラおよびアラームハンドラとしては登録できません。**

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"
```

4. **周期ハンドラおよびアラームハンドラはシステムクロックの割り込みハンドラからサブルーチン呼び出しにより起動されます。**

```
#include <itron.h>
#include <kernel.h>
#include "kernel_id.h"

void cychand(VP_INT inf)
{
        /* 処理 */
}
```

図 5.5　C 言語で記述した周期ハンドラの例

[48] ハンドラと関数名との対応は、コンフィギュレーションファイルにより行います。

5.2 アセンブリ言語によるコーディング方法

本節では、アセンブリ言語を用いてアプリケーションを記述する方法について述べます。

5.2.1 タスクの記述方法

アセンブリ言語を用いてタスクを記述する場合、以下の項目に注意してください。

1. **ファイルの先頭で必ず "mr308.inc"をインクルードしてください。**
2. **タスクの開始アドレスを示すシンボルは外部シンボル宣言[49]をおこなってください。**
3. **タスクは無限ループか ext_tsk サービスコールで終了してください。**

```
        .INCLUDE mr308.inc ----- (1)
        .GLB    task    ----- (2)

task:
                ; 処理
        jmp   task      ----- (3)
```

図 5.6　アセンブリ言語で記述した無限ループタスクの例

```
        .INCLUDE mr308.inc
        .GLB    task

task:
                ; 処理
        ext_tsk
```

図 5.7　アセンブリ言語で記述した ext_tsk で終了するタスクの例

4. **タスク起動時のレジスタの初期値は、R0、PC、SB、FLG レジスタ以外は不定です。**
5. **タスクを指定する場合はコンフィギュレーションファイルのタスク定義の項目”name”に記述した文字列で指定してください**

```
        wup_tsk  #ID_task
```

6. **イベントフラグ、セマフォ、メールボックスを指定する場合は、コンフィギュレーションファイルで定義したそれぞれの名前の文字列で指定してください。**

   例えば、 コンフィギュレーションファイルで以下のようにセマフォを定義した場合、

```
semaphore[1]{
        name            = ID_abc;
};
```

   このセマフォを指定するには以下のようにおこなってください。

```
sig_sem  #ID_abc
```

7. **周期ハンドラ、アラームハンドラを指定する場合は、コンフィギュレーションファイルのアラームハンドラもしくは周期ハンドラ定義の項目”name”に記述した文字列で指定してください。**

```
sta_cyc  #ID_cyc
```

8. **MR308 システム起動時に起動されるタスクは、コンフィギュレーションファイルで設定します。[50]**

[49] .GLB 指示命令を使用してください。
[50] このタスク ID 番号とタスク(プログラム)との対応づけは、コンフィギュレーションファイルにより行います。

5.2.2 カーネル管理(OS 依存)割り込みハンドラの記述方法

アセンブリ言語を用いてカーネル管理(OS 依存)割り込みハンドラを記述する場合、以下の項目に注意してください。

1. **ファイルの先頭で必ず"mr308.inc"をインクルードしてください。**
2. **割り込みハンドラの開始アドレスを示すシンボルは外部宣言（グローバル宣言)[51]をおこなってください。**
3. **ハンドラ内で使用するレジスタは、 ハンドラの入口でセーブし、使用後復帰して下さい。**
4. **ret_int サービスコールにて復帰してください。また、割り込みハンドラ関数から呼び出した関数のなかで割り込みハンドラを終了することはできません。**

```
                .INCLUDE mr308.inc              ------(1)
                .GLB    inth                    ------(2)

inth:
                ;  使用レジスタ退避                        ------(3)
                iwup_tsk  #ID_task1
                        :
                        処     理
                        :

                ;  使用レジスタ復帰                        ------(3)

                ret_int                         ------(4)
```

図 5.8　カーネル管理(OS 依存)割り込みハンドラの例

[51] .GLB 指示命令を使用してください。

5.2.3 カーネル管理外(OS 独立)割り込みハンドラ記述方法

1. **割り込みハンドラの開始アドレスを示すシンボルは外部宣言（グローバル宣言）して下さい。**
2. **ハンドラ内で使用するレジスタは入り口でセーブし、使用後復帰して下さい。**
3. **REIT 命令で終了してください。**
4. **カーネル管理外(OS 独立)割り込みハンドラからは、サービスコールは発行できません。**
   (注)サービスコールを発行した場合は不正動作をするので十分注意して下さい。

5. **カーネル管理外(OS 独立)割り込みハンドラ内で多重割り込みを許可する場合は、カーネル管理外(OS 独立)割り込みハンドラの割り込み優先レベルは、他のカーネル管理(OS 依存)割り込みハンドラの割り込み優先レベルより必ず高くしてください。[52]**

```
        .GLB    inthand                 ----- (1)

inthand:
        ; 使用レジスタ退避 ----- (2)
        ; 割り込み処理
        ; 使用レジスタ復帰          ----- (2)
        REIT                            ----- (3)
```

図 5.9 カーネル管理外(OS 独立)割り込みハンドラの例

[52] カーネル管理外(OS 独立)割り込みハンドラの割り込み優先レベルを、カーネル管理(OS 依存)割り込みハンドラの割り込み優先レベルより低くしたい場合は、カーネル管理外(OS 独立)割り込みハンドラをカーネル管理(OS 依存)割り込みハンドラの記述に変更してください。

5.2.4 周期ハンドラ、アラームハンドラの記述方法

アセンブリ言語を用いて周期ハンドラおよびアラームハンドラを記述する場合、以下の点に注意してください。

1. **ファイルの先頭で必ず"mr308.inc"をインクルードしてください。**
2. **ハンドラの開始アドレスを示すシンボルは外部宣言（グローバル宣言）[53]をおこなってください。**
3. **周期ハンドラ、アラームハンドラは全て RTS 命令（サブルーチンリターン命令）にて復帰してください。**

例えば、

```
.INCLUDE        mr308.inc       ----- (1)
.GLB            cychand         ----- (2)

cychand:
                :
        ; ハンドラ処理
                :

        rts                     ----- (3)
```

図 5.10 アセンブリ言語で記述したハンドラの例

[53] .GLB 指示命令を使用してください。

5.3 INT 命令の使用について

MR308 では、INT 命令の割り込み番号を表 5.2に示すようにサービスコール発行のため予約しています。そのため、ユーザーアプリケーションでソフトウェア割り込みを使用する場合は、63～58,55 以外の割り込みを使用して下さい。

表 5.2　割り込み番号の割り当て

割り込み番号	使用するサービスコール
63	タスクコンテキストからのみ発行可能なサービスコール
62	非タスクコンテキストからのみ発行可能なサービスコール タスクコンテキスト､非タスクコンテキストの両方から発行可能なサービスコール
61	ret_int サービスコール
60	dis_dsp サービスコール
59	loc_cpu, iloc_cpu サービスコール
58	ext_tsk サービスコール
55	tsnd_dtq, twai_flg, vtsnd_dtq サービスコール

5.4 レジスタバンクについて

MR308 では、タスク起動時のコンテキストは、レジスタバンク 0 を使用しています。
カーネル処理中にレジスタバンク切り替えは行いません。
プログラム誤動作の原因となりますので、以下の点にご注意ください。

- タスク内では、レジスタバンク切り替えは行わないで下さい。
- レジスタバンク切り替えを指定している割り込み同士が、多重に割り込まないようにして下さい。

5.5 割り込みについて

5.5.1 割り込みハンドラの種類

MR308 の割り込みハンドラには、カーネル管理(OS 依存)割り込みハンドラとカーネル管理外(OS 独立)割り込みハンドラを定義しています。それぞれの割り込みハンドラの定義を以下に示します。

- カーネル管理(OS 依存)割り込みハンドラ
  以下の 2 つの条件のうち、どちらか一方を満たすものをカーネル管理(OS 依存)割り込みハンドラとして定義します。
  - ♦ サービスコールを発行する割り込みハンドラ
  - ♦ サービスコールを発行する割り込みハンドラが多重ではいる割り込みハンドラ

  カーネル管理(OS 依存)割り込みハンドラの IPL 値は、カーネル割り込みマスクレベル(OS 割り込み禁止レベル) (system.IPL)以下 (IPL=0 ～ system.IPL)[54]にする必要があります。

- カーネル管理外(OS 独立)割り込みハンドラ
  以下の 2 つの条件の両方を満たすものをカーネル管理外(OS 独立)割り込みハンドラとして定義します。
  - ♦ サービスコールを発行しない割り込みハンドラ
  - ♦ サービスコールを発行する割り込みハンドラ (システムクロック割り込みハンドラ)が多重割り込みではいらないもの

  カーネル管理外(OS 独立)割り込みハンドラの IPL 値は、 (system.IPL+1) ～ 7 にする必要があります。すなわち、カーネル管理外(OS 独立)割り込みハンドラの IPL 値を OS 割り込み禁止レベル以下に設定できません。

図 5.11に、カーネル割り込みマスクレベル(OS 割り込み禁止レベル)を 3 にした場合の、カーネル管理(OS 依存)割り込みハンドラとカーネル管理外(OS 独立)割り込みハンドラの関係を示します。

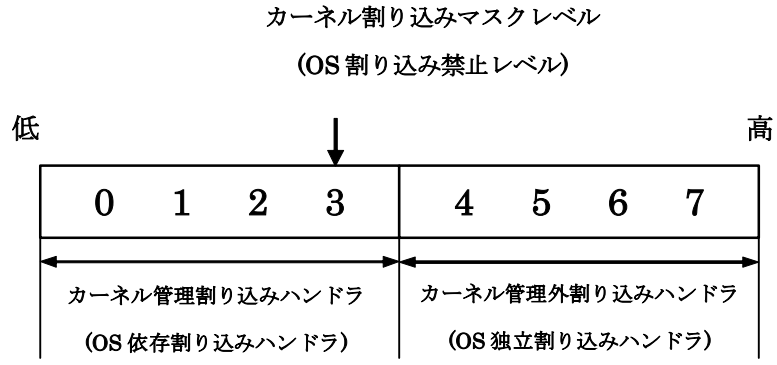


図 5.11 割り込みハンドラの IPL

5.5.2 ノンマスカブル割り込みについて

NMI 割り込みおよび監視タイマ割り込みは、必ず、カーネル管理外(OS 独立)割り込みにしてください。カーネル管理(OS 依存)割り込みにした場合、プログラム誤動作の原因となりますので、ご注意ください。

[54] system.IPL は、コンフィギュレーションファイルで設定します。

5.5.3 割り込み制御方法

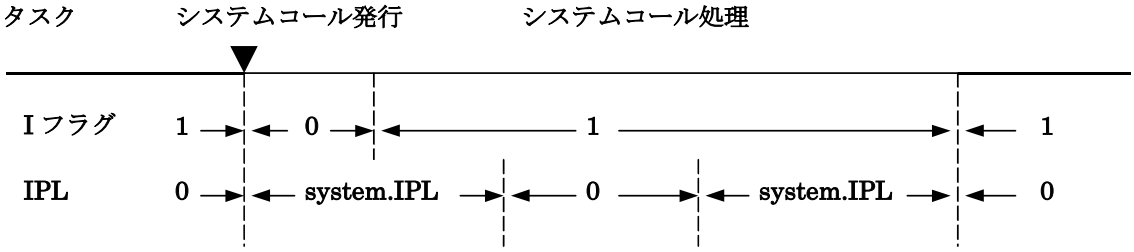
サービスコール内の割り込み禁止/許可の制御は、IPL の操作により行っています。

サービスコール内での IPL 値は、カーネル割り込みマスクレベル(OS 割り込み禁止レベル) (system.IPL)にして、カーネル管理(OS 依存)割り込みハンドラの割り込みを禁止しています。全ての割り込みを許可できる箇所では、サービスコール発行時の IPL 値に戻します。

図 5.12に、サービスコール内での割り込み許可フラグと IPL の状態を示します。

- タスクコンテキストからのみ発行できるサービスコールの場合

・システムコール発行前の I フラグが 1 の場合



・システムコール発行前の I フラグが 0 の場合

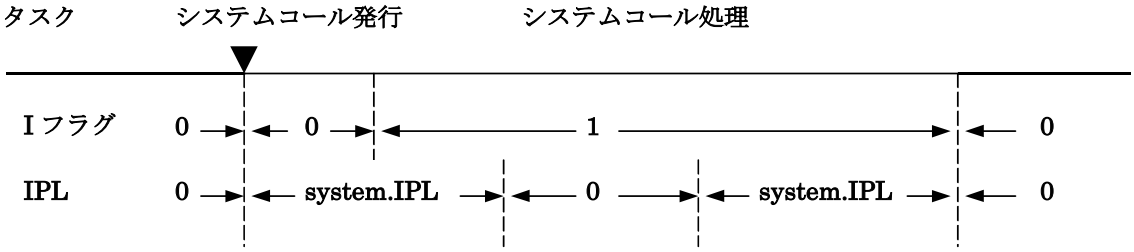
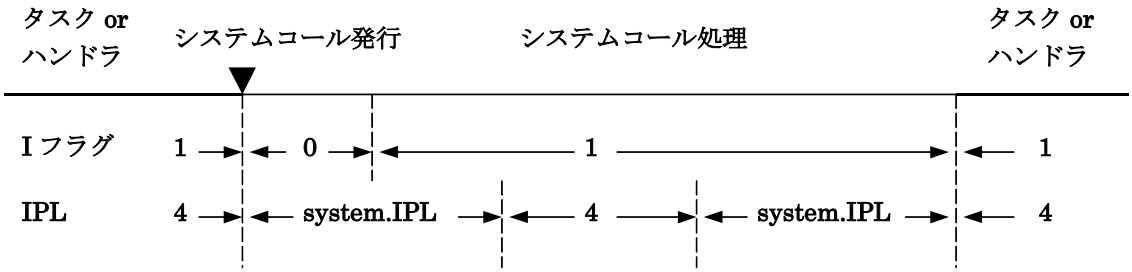


図 5.12 タスクコンテキストからのみ発行できるサービスコール内での割り込み制御

- 非タスクコンテキストからのみ発行できるサービスコール、もしくは、タスクコンテキストと非タスクコンテキストの両方から発行できるサービスコールの場合

・システムコール発行前のＩフラグが 1 の場合



・システムコール発行前のＩフラグが 0 の場合

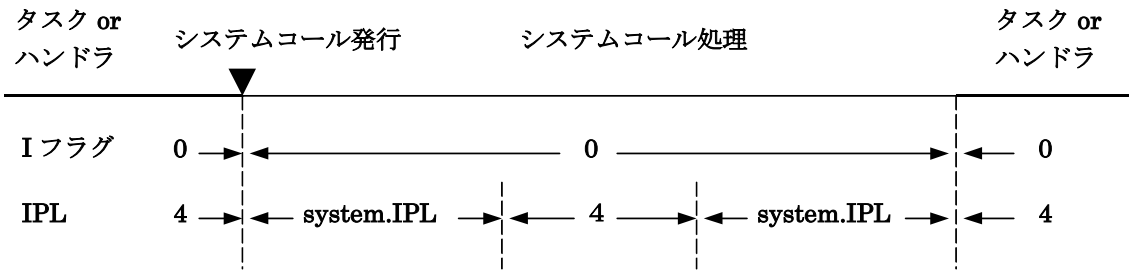


図 5.13 非タスクコンテキストから発行できるサービスコール内での割り込み制御

図 5.12、図 5.13に示すように割り込み許可フラグおよび IPL は、サービスコール内で変化します。そのため、ユーザーアプリケーション内で割り込みを禁止したい場合、割り込み許可フラグおよび IPL の操作によって割り込みを禁止にする方法をとらないで下さい。

割り込みの制御は、以下に示す二つの方法で実現して下さい。

1. 禁止にしたい割り込みの割り込み制御レジスタ（SFR)を変更する。

2. loc_cpu,iloc_cpu ～ unl_cpu,iunl_cpu を使用する。

loc_cpu,iloc_cpu サービスコールにより、制御できる割り込みは、カーネル管理(OS 依存)割り込みのみです。カーネル管理外(OS 独立)割り込みを制御する場合には、1による方法で行って下さい。

5.6 ディスパッチ遅延について

MR308 では、ディスパッチ遅延に関するサービスコールが 4 つあります。

- dis_dsp
- ena_dsp
- loc_cpu,iloc_cpu
- unl_cpu,iunl_cpu

これらのサービスコールを使用し、一時的にディスパッチを遅延した場合のタスクの扱いについて以下に記述します。

1. ディスパッチ遅延中の実行タスクがプリエンプトされる場合

ディスパッチが禁止されている間は、実行中のタスクがプリエンプトされるべき状況となっても、新たに実行すべき状態となったタスクにはディスパッチされません。実行するべきタスクへのディスパッチは、ディスパッチ禁止状態が解除されるまで遅延されます。ディスパッチ遅延中は、

- 実行中のタスクは RUNNING 状態であり、レディキューにつながれている。
- ディスパッチ禁止解除後に実行するタスクは、READY 状態であり、(タスクがつながれている中で)最高優先度のレディキューにつながれている。

2. ディスパッチ遅延中の isus_tsk,irsm_tsk

また、ディスパッチ禁止状態で起動された割り込みハンドラから、実行中のタスク（dis_dsp を発行したタスク）に対して isus_tsk を発行し SUSPENDED 状態へ移行させようとした場合、タスク状態の遷移はディスパッチ禁止状態が解除されるまで遅延されます。ディスパッチ遅延中は、

- 実行中のタスクの状態の扱いは、OS 内部では、ディスパッチ遅延解除後の状態として扱います。そのため、実行中のタスクに対して発行された isus_tsk では、実行中のタスクをレディキューからはずし、SUSPENDED 状態に移行します。エラーコードは E_OK を返します。この後、実行中のタスクに対して irsm_tsk が発行されると、実行中のタスクをレディキューにつなぎ、エラーコードは E_OK を返します。ただし、ディスパッチ遅延が解除されるまでタスクの切り替えは起こりません。
- ディスパッチ禁止解除後に実行するタスクは、レディキューにつながれています。

3. ディスパッチ遅延中の rot_rdq, irot_rdq

ディスパッチ遅延中に、rot_rdq(TPRI_RUN=0)を発行した場合、自タスクの持つ優先度のレディキューを回転させます。また、irot_rdq(TPRI_RUN=0)を発行した場合、実行中のタスクが持つ優先度のレディキューが回転します。この場合、実行中のタスクは該当レディキューにはつながれていない場合があります。（ディスパッチ遅延中に、実行タスクに対し isus_tsk が発行された場合など。）

4. 注意事項

- dis_dsp により、ディスパッチが禁止されている状態で、自タスクを待ち状態に移す可能性のあるサービスコール（slp_tsk,wai_sem など)は発行しないで下さい。
- loc_cpu により CPU ロック状態で ena_dsp,dis_dsp は発行できません。
- dis_dsp を何回か発行して、その後、ena_dsp を 1 回発行しただけでディスパッチ禁止状態は解除されます。

5.7 初期起動タスクについて

MR308 では、システム起動時に READY 状態からスタートするタスクを指定できます。つまり、タスク属性として TA_STA を付加します。この指定はコンフィギュレーションファイルで設定を行います。

設定方法の詳細については、- 105 -ページを参照して下さい。

5.8 MR308 スタートアッププログラムの修正方法

MR308 には、以下に示す 2 種類のスタートアッププログラムが用意されています。

- start.a30
  アセンブリ言語を使って、プログラムを作成した時に使用するスタートアッププログラムです。

- crt0mr.a30
  C 言語を使って、プログラムを作成した時に使用するスタートアッププログラムです。
  "start.a30"に C 言語の初期化ルーチンを追加したものです。

スタートアッププログラムは以下のようなことを行っています。

- リセット後のプロセッサの初期化
- C 言語の変数の初期化（crt0mr.a30 のみ）
- システムタイマの設定
- MR308 のデータ領域の初期化

このスタートアッププログラムは、環境変数 "LIB308"の示すディレクトリからカレントディレクトリへコピーして下さい。なお、必要があれば以下の示す箇所を修正、あるいは追加して下さい。
HEW を使用する際は、プロジェクト生成時に自動的にサンプルを生成するか、既に作成したものを利用するか選択することが出来ます。

- プロセッサモードレジスタの設定
  プロセッサモードレジスタに、システムに合わせたプロセッサモードを設定して下さい。

- ユーザーで必要な初期化プログラムの追加
  ユーザーに必要な初期化プログラムを追加する場合は、C 言語用スタートアッププログラム(crt0mr.a30)の標準入出力関数の初期化の直前に追加して下さい。
  標準入出力関数を使用しない場合は該当部分をコメントにしてください。

5.8.1 C 言語用スタートアッププログラム (crt0mr.a30)

図 5.14に C 言語用スタートアッププログラム crt0mr.a30 を示します。

```
 1 ;***********************************************************************
 2 ;
 3 ;       MR308 start up program for C language
 4 ;       COPYRIGHT(C) 2003 RENESAS TECHNOLOGY CORPORATION
 5 ;       AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
 6 ;       MR308 V.1.1.0 Release 1
 7 ;
 8 ; ***********************************************************************
 9 ;         "$Id: crt0mr.a30,v 1.1 2005/05/20 06:28:47 inui Exp $"
10 ;*A1* 2005-02-28 for ES
11 ;
12        .LIST   OFF
13        .INCLUDE        c_sec.inc
14        .INCLUDE        mr308.inc
15        .INCLUDE        sys_rom.inc
16        .INCLUDE        sys_ram.inc
17        .LIST   ON
18
19        .GLB    __SYS_INITIAL
20        .GLB    __END_INIT
21        .GLB    __init_sys,__init_tsk
22
23        .IF             M16C70!=0
24 regoffset       .EQU    -0220H
25        .ELSE
26 regoffset       .EQU    0
27        .ENDIF
28
29 ;-----------------------------------------------------------------------
30 ; SBDATA area definition
31 ;-----------------------------------------------------------------------
32        .GLB    __SB__
33        .SB     __SB__
34
35 ;=======================================================================
36 ; Initialize Macro declaration
37 ;-----------------------------------------------------------------------
38 N_BZERO .MACRO  TOP_,SECT_
39        MOV.B   #00H, R0L
40        MOV.L   #TOP_, A1
41        MOV.W   #sizeof SECT_, R3
42        SSTR.B
43        .ENDM
44
45 N_BCOPY .MACRO  FROM_,TO_,SECT_
46        MOV.L   #FROM_,A0
47        MOV.L   #TO_,A1
48        MOV.W   #sizeof SECT_, R3
49        SMOVF.B
50        .ENDM
51
52 BZERO   .MACRO  TOP_,SECT_
53        .local  _end, _loop
54
55        MOV.L   #TOP_, A1
56        MOV.B   #00H, R0L
57        MOV.L   #(sizeof SECT_ & 0FFFFFFH), R3R1
58        XCHG.W  R1,R3
59 _loop:
60        SSTR.B
61        CMP.W   #0,R1
62        JEQ     _end
63        MOV.B   R0L,[A1]
64        ADD.L   #1,A1
65        MOV.W   #0FFFFH,R3
66        SUB.W   #1,R1
67        JMP     _loop
68 _end:
69        .ENDM
70
71 BCOPY   .MACRO  FROM_,TO_,SECT_
72        .local  _end, _loop
73
74        MOV.L   #FROM_,A0
75        MOV.L   #TO_,A1
```

```
 76        MOV.L   #(sizeof SECT_ & 0FFFFFFH), R3R1
 77        XCHG.W  R1,R3
 78 _loop:
 79        SMOVF.B
 80        CMP.W   #0,R1
 81        JEQ     _end
 82        MOV.B   [A0],[A1]
 83        ADD.L   #1,A1
 84        ADD.L   #1,A0
 85        MOV.W   #0FFFFH,R3
 86        SUB.W   #1,R1
 87        JMP     _loop
 88 _end:
 89        .ENDM
 90
 91 ;=====================================================================
 92 ; Interrupt section start
 93 ;---------------------------------------------------------------------
 94        .SECTION       MR_KERNEL,CODE,ALIGN
 95
 96 ;---------------------------------------------------------------------
 97 ; after reset,this program will start
 98 ;---------------------------------------------------------------------
 99 __SYS_INITIAL:
100        LDC     #__Sys_Sp,ISP   ; set initial ISP
101
102        MOV.B   #2,0AH
103        MOV.B   #00,PMOD              ; Set Processor Mode Register
104        MOV.B   #0,0AH
105        LDC     #0010H,FLG
106        LDC     #__SB__,SB
107        LDC     #0000H,FLG
108        LDC     #__Sys_Sp,FB
109        LDC     #__SB__,SB
110
111 ; +-------------------------------------------------------+
112 ; |     ISSUE SYSTEM CALL DATA INITIALIZE                 |
113 ; +-------------------------------------------------------+
114        ; For PD308
115        __INIT_ISSUE_SYSCALL
116
117 ;=====================================================================
118 ; MR_RAM zero clear
119 ;---------------------------------------------------------
120        N_BZERO MR_RAM_NE_top,MR_RAM_NE
121        N_BZERO MR_RAM_NO_top,MR_RAM_NO
122        BZERO   MR_RAM_top,MR_RAM
123
124 ;=====================================================================
125 ; NEAR area initialize.
126 ;---------------------------------------------------------
127 ; bss zero clear
128 ;---------------------------------------------------------
129        N_BZERO bss_SE_top,bss_SE
130        N_BZERO bss_SO_top,bss_SO
131
132        N_BZERO bss_NE_top,bss_NE
133        N_BZERO bss_NO_top,bss_NO
134
135 ;---------------------------------------------------------
136 ; initialize data section
137 ;---------------------------------------------------------
138        N_BCOPY data_SEI_top,data_SE_top,data_SE
139        N_BCOPY data_SOI_top,data_SO_top,data_SO
140        N_BCOPY data_NEI_top,data_NE_top,data_NE
141        N_BCOPY data_NOI_top,data_NO_top,data_NO
142
143 ;=====================================================================
144 ; FAR area initialize.
145 ;---------------------------------------------------------
146 ; bss zero clear
147 ;---------------------------------------------------------
148        BZERO   bss_FE_top,bss_FE
149        BZERO   bss_FO_top,bss_FO
150
151 ;---------------------------------------------------------
152 ; Copy edata_E(O) section from edata_EI(OI) section
153 ;---------------------------------------------------------
154        BCOPY   data_FEI_top,data_FE_top,data_FE
155        BCOPY   data_FOI_top,data_FO_top,data_FO
```

```
156
157         LDC     #__Sys_Sp,SP
158         LDC     #__Sys_Sp,FB
159
160
161 ;---------------------------------------------------------------
162 ; Set System IPL and Set Interrupt Vector
163 ;---------------------------------------------------------------
164         MOV.B   #0,R0L
165         MOV.B   #__SYS_IPL,R0H
166         LDC     R0,FLG
167         LDC     #__INT_VECTOR,INTB
168
169 ; +-----------------------------------------------------+
170 ; |     System timer interrupt setting                  |
171 ; +-----------------------------------------------------+
172     .IF USE_TIMER
173         MOV.B   #stmr_mod_val,stmr_mod_reg+regoffset    ; set timer mode
174         MOV.W   #stmr_cnt,stmr_ctr_reg+regoffset        ; set interval count
175         MOV.B   #stmr_int_IPL,stmr_int_reg              ; set timer IPL
176         OR.B    #stmr_bit+1,stmr_start+regoffset        ; system timer start
177     .ENDIF
178
179 ; +-----------------------------------------------------+
180 ; |     System timer initialize                         |
181 ; +-----------------------------------------------------+
182     .IF USE_SYSTEM_TIME
183         MOV.W   #__D_Sys_TIME_L,__Sys_time+4
184         MOV.W   #__D_Sys_TIME_M,__Sys_time+2
185         MOV.W   #__D_Sys_TIME_H,__Sys_time
186     .ENDIF
187
188 ; +-----------------------------------------------------+
189 ; |     User Initial Routine ( if there are )           |
190 ; +-----------------------------------------------------+
191 ; Initialize standard I/O
192         .GLB    _init
193         JSR.A   _init
194
195 ; +-----------------------------------------------------+
196 ; |     Initalization of System Data Area               |
197 ; +-----------------------------------------------------+
198         JSR.W   __init_sys
199         JSR.W   __init_tsk
200
201     .IF __MR_TIMEOUT
202         .GLB    __init_tout
203         JSR.W   __init_tout
204     .ENDIF
205
206     .IF __NUM_FLG
207         .GLB    __init_flg
208         JSR.W   __init_flg
209     .ENDIF
210
211     .IF __NUM_SEM
212         .GLB    __init_sem
213         JSR.W   __init_sem
214     .ENDIF
215
216     .IF   __NUM_DTQ
217  .GLB     __init_dtq
218  JSR.W    __init_dtq
219     .ENDIF
220
221     .IF   __NUM_VDTQ                        ;*A1*
222  .GLB     __init_vdtq
223  JSR.W    __init_vdtq
224     .ENDIF
225
226     .IF __NUM_MBX
227         .GLB    __init_mbx
228         JSR.W   __init_mbx
229     .ENDIF
230
231     .IF ALARM_HANDLER
232         .GLB    __init_alh
233         JSR.W   __init_alh
234     .ENDIF
235
```

```
236     .IF CYCLIC_HANDLER
237        .GLB    __init_cyh
238        JSR.W   __init_cyh
239     .ENDIF
240
241     .IF   __NUM_MPF                         ;*A1*
242  ; Fixed Memory Pool
243  .GLB     __init_mpf
244  JSR.W    __init_mpf
245     .ENDIF
246
247     .IF   __NUM_MPL                         ;*A1*
248  ; Variable Memory Pool
249  .GLB     __init_mpl
250  JSR.W    __init_mpl
251     .ENDIF
252
253        ; For PD308
254        __LAST_INITIAL
255
256 __END_INIT:
257
258 ; +-----------------------------------------------------+
259 ; |       Start initial active task                     |
260 ; +-----------------------------------------------------+
261        __START_TASK
262
263        .GLB    __rdyq_search
264        JMP.W   __rdyq_search
265
266 ; +-------------------------------------------+
267 ; |    Define Dummy                           |
268 ; +-------------------------------------------+
269        .GLB    __SYS_DMY_INH
270 __SYS_DMY_INH:
271        REIT
272
273 .IF    CUSTOM_SYS_END
274 ; +-------------------------------------------+
275 ; | Syscall exit rouitne to customize
276 ; +-------------------------------------------+
277        .GLB    __sys_end
278 __sys_end:
279        ; Customize here.
280        REIT
281 .ENDIF
282
283 ; +-------------------------------------------+
284 ; |           exit() function                 |
285 ; +-------------------------------------------+
286        .GLB    _exit,$exit
287 _exit:
288 $exit:
289        JMP     _exit
290
291 .IF USE_TIMER
292 ; +-------------------------------------------+
293 ; |      System clock interrupt handler       |
294 ; +-------------------------------------------+
295        .GLB           __SYS_STMR_INH
296        .ALIGN
297 __SYS_STMR_INH:
298        ; process issue system call
299        ; For PD308
300        __ISSUE_SYSCALL
301
302        ; System timer interrupt handler
303        _STMR_hdr
304
305        ret_int
306 .ENDIF
307
308        .END
309
310 ; ************************************************************
311 ;      COPYRIGHT(C) 2003 RENESAS TECHNOLOGY CORPORATION
312 ;      AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
313 ; ************************************************************
```

図 5.14 C 言語用スタートアッププログラム（crt0mr.a30）

以下に標準スタートアッププログラム crt0mr.a30 の内容について説明します。

1. **セクション定義ファイルを組み込みます。［図 5.14の 13 行目］**
2. **MR308 用インクルードファイルを組み込みます。［図 5.14の 14 行目］**
3. **システム ROM 領域定義ファイルを組み込みます。［図 5.14の 15 行目］**
4. **システム RAM 領域定義ファイルを組み込みます。［図 5.14の 16 行目］**
5. **リセット直後に起動される初期化プログラム__SYS_INITIAL です。［図 5.14の 99 行目-256 行目］**
   - システムスタックポインタの設定［図 5.14の 100 行目］
   - プロセッサモードレジスタの設定［図 5.14の 102 行目-104 行目］
   - FLG、SB、FB レジスタの設定［図 5.14の 105 行目-109 行目］
   - C 言語の初期設定をおこないます。［図 5.14の 129 行目-155 行目］
   - カーネルマスク(OS 割り込み禁止)レベルの設定［図 5.14の 164 行目-166 行目］
   - 割り込みベクタテーブルのアドレス設定［図 5.14の 167 行目］
   - MR308 のシステムクロック割り込みの設定をおこないます。［図 5.14の 172 行目-177 行目］
   - 標準入出力関数の初期化[図 5.14の 192 行目-193 行目］
     標準入出力関数を使用しない場合は、この行をコメントアウトしてください。
   - MR308 のシステム時刻の初期設定をおこないます。［図 5.14の 182 行目-186 行目］
6. **必要があればアプリケーション固有の初期設定をおこないます。［図 5.14の 191 行目］**
7. **MR308 が使用する RAM データの初期化をおこないます。［図 5.14の 198 行目-251 行目］**
8. **スタートアップの終了を示すビットをセットします。［図 5.14の 254 行目］**
9. **初期起動タスクを起動します。［図 5.14の 261 行目-265 行目］**
10. **システムクロックの割り込みハンドラです。［図 5.14の 295 行目-306 行目］**

5.9 メモリ配置方法

アプリケーションプログラムのデータのメモリ配置方法について説明します。
メモリ配置を設定するためには、MR308 が提供しているセクションファイルで設定します。
MR308 では、以下に示す 2 種類のセクションファイルが用意されています。

- asm_sec.inc
  アセンブリ言語で、アプリケーション開発を行った場合に使用します。
  各セクションの詳細については、- 91 -ページを参照して下さい。

- c_sec.inc
  C 言語で、アプリケーション開発を行った場合に使用します。
  c_sec.inc は、"asm_sec.inc"に C コンパイラ NC308WA が生成するセクションを追加したものです。
  各セクションの詳細については、- 92 -ページを参照して下さい。

ユーザーシステムに合わせて、セクション配置、開始アドレスの設定を変更して下さい。
セクションファイルの変更方法を以下に示します。

例
プログラムセクションの先頭をF0000H**番地から**F1000H**番地に変更したい場合**

```
.section        program
.org    0F0000H ; この部分を F1000H に修正

        ↓

.section        program

.org    0F1000H ;
```

5.9.1 start.a30 のセクションの配置

アセンブリ言語用サンプルスタートアッププログラム"start.a30"のセクション配置は、"asm_sec.inc"で定義しています。
セクションの再配置が必要な場合は、"asm_sec.inc"を編集してください。
以下に、サンプルセクション定義ファイル"asm_sec.inc"で定義している各セクションについて説明します。

- MR_RAM_DBG セクション
  MR308 のデバッグ機能に必要な RAM データが入るセクションです。
  このセクションは必ず内蔵 RAM 領域に配置して下さい。
- MR_RAM_NE セクション
- MR_RAM_NO セクション
  MR308 のシステム管理データで、アブソリュートアドレシングで参照する RAM データが入るセクションです。
  このセクションは必ず 0～0FFFFH(near 領域)以内に配置して下さい。
- MR_RAM セクション
  MR308 のシステム管理データで、アブソリュートアドレシングで参照する RAM データが入るセクションです。
- stack セクション
  各タスクのデフォルトのユーザースタック、およびシステムスタックのセクションです。
- MR_HEAP セクション
  可変長メモリプール、固定長メモリプール、データキュー領域が格納されるセクションです。
- MR_KERNEL セクション
  MR308 カーネルプログラムを格納するセクションです。
- MR_CIF セクション
  MR308 インターフェイスライブラリを格納するセクションです。
- MR_ROM セクション
  MR308 カーネルが参照するタスクの開始番地などのデータを格納するセクションです。
- program セクション
  ユーザープログラムを格納するセクションです。
  このセクションは MR308 カーネルは全く使用していません。したがって、ユーザーで自由に使用することができます。
- program_S セクション
  スペシャルページ呼び出しの関数を格納するセクションです。
  このセクションは MR308 カーネルは全く使用していません。
- fvector セクション
  スペシャルページ呼び出しのベクタアドレスを格納するセクションです。
  このセクションは MR308 カーネルは全く使用していません。
- INTERRUPT_VECTOR セクション
- FIX_INTERRUPT_VECTOR セクション
  割り込みベクタを格納するセクションです。このセクションの開始番地は使用する M16C ファミリ機種により異なります。サンプルスタートアッププログラムの番地は M16C/80 シリーズ用のものです。他のグループを使用する場合は変更してください。

5.9.2 crt0mr.a30 のセクション配置

C 言語用サンプルスタートアッププログラム"crt0mr.a30"のセクション配置は、"c_sec.inc"で定義しています。
セクションの再配置が必要な場合は、"c_sec.inc"を編集してください。
サンプルセクション定義ファイル"c_sec.inc"で定義しているセクションは、アセンブリ言語用スタートアッププログラムのセクション配置"asm_sec.inc"に、以下のセクションを定義したものです。

- data_SE セクション
- bss_SE セクション
- data_SO セクション
- bss_SO セクション
- data_NE セクション
- bss_NE セクション
- data_NO セクション
- bss_NO セクション
- rom_NE セクション
- rom_NO セクション
- data_FE セクション
- bss_FE セクション
- data_FO セクション
- bss_FO セクション
- rom_FE セクション
- rom_FO セクション
- data_SEI セクション
- data_SOI セクション
- data_NEI セクション
- data_NOI セクション
- data_FEI セクション
- data_FOI セクション

  これらのセクションは、NC308WA が生成するセクションです。アセンブリ言語用セクションファイルでは、このセクションは定義されていません。
  詳細は、NC308WA のマニュアルを参照してください。

サンプルスタートアッププログラムのセクションの配置を図 5.15に示します。

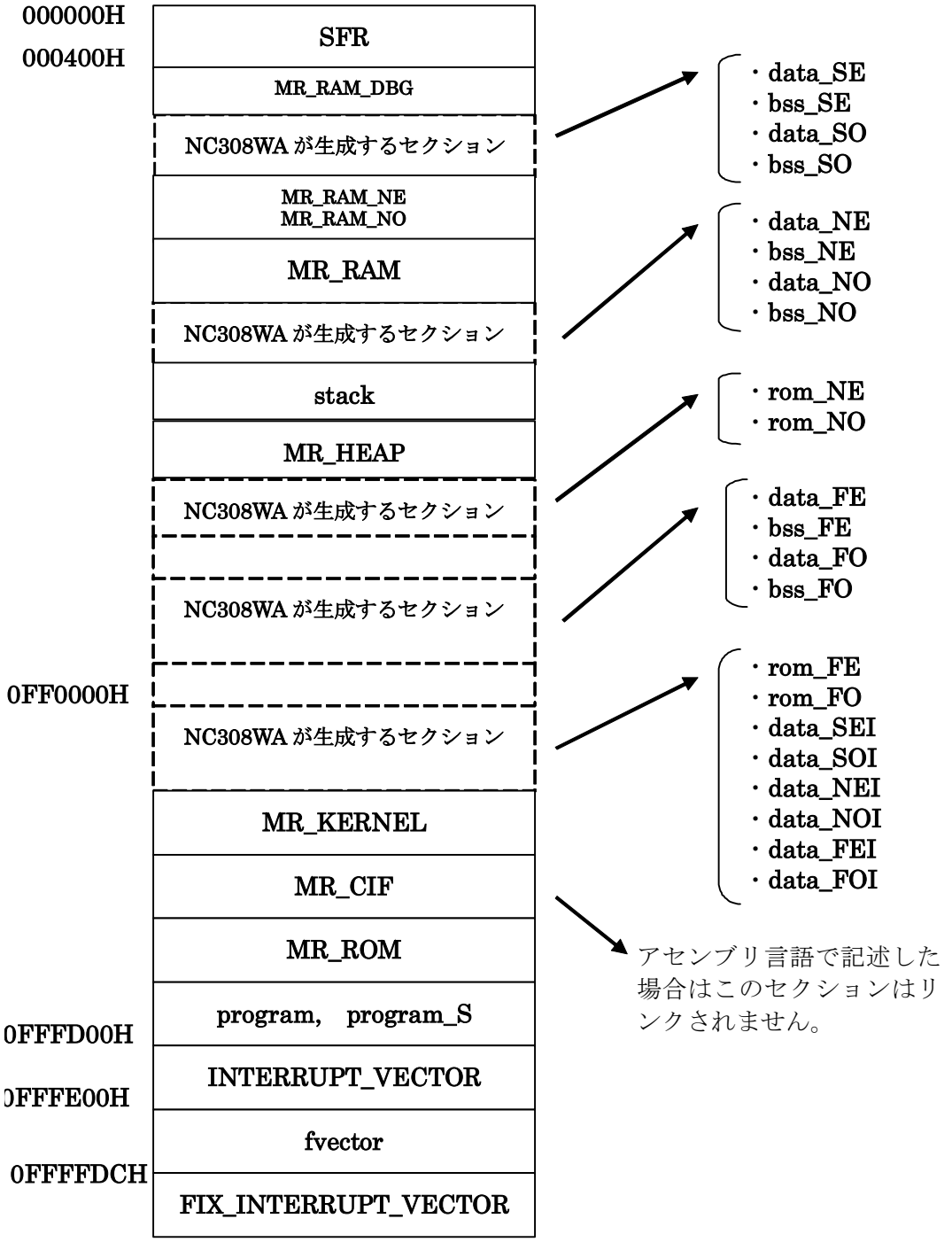


図 5.15 C 言語スタートアッププログラムのセクション配置

5.10 M16C/70 シリーズで使用する場合について

M16C/70 シリーズを使用する場合、TM やコンフィギュレータが出力する makefile でコンパイル時に以下のオプションを必ず設定してください。
オプションの詳細な説明は各ツールのマニュアルを参照ください。

《NC308 コンパイルオプション》

```
-D M16C70=1
```

《AS308 コンパイルオプション》

```
-DM16C70=1
```

第 6 章　　コンフィギュレータの使用方法

6.1 コンフィギュレーションファイルの作成方法

アプリケーションプログラムのコーディング、スタートアッププログラムの修正が終わると、そのアプリケーションプログラムを MR308 システムに登録する必要があります。これをおこなうのがコンフィギュレーションファイルです。

6.1.1 コンフィギュレーションファイル内の表現形式

この節ではコンフィギュレーションファイル内における定義データの表現形式について説明します。

コメント文

'//'から行の終わりまではコメント文とみなし、処理の対象になりません。

文の終わり

';'で文を終ります。

数値

数値は以下の形式で入力できます。

1. **16 進数**

   数値の先頭に'0x'か'0X'を付加します。または、数値の最後に'h'か'H'を付加します。後者の場合でかつ先頭が英文字（A～F)で始まる場合は先頭に必ず'0'を付加してください。なおここで使用する数値表現で英文字（A～F)は大文字・小文字を識別しません。[55]

2. **10 進数**

   23 のように整数のみで表します。ただし'0'で始めることはできません。

3. **8 進数**

   数値の先頭に'0'を付加するか数値の最後に'O'もしくは'o'を付加します。

4. **2 進数**

   数値の最後に'B'または'b'を付加します。ただし'0'で始めることはできません。

表 6.1　数値表現例

16 進数	0xf12 0Xf12 0a12h 0a12H 12h 12H
10 進数	32
8 進数	017 17o 170
2 進数	101110b 101010B

また数値内に演算子を記述できます。使用できる演算子を表 6.2に示します。

[55] 数値表現内の'A'～'F','a'～'f'を除いて全ての文字は、大文字・小文字の区別を行います。

表 6.2　演算子

演算子	優先度	演算方向
()	高	左から右
- (単項マイナス)		右から左
* / %		左から右
+ - (二項マイナス)	低	左から右

数値の例を以下に示します。

- 123
- 123 + 0x23
- (23/4 + 3) * 2
- 100B + 0aH

シンボル

シンボルは数字、英大文字、英小文字、'_'（アンダースコア）、'?'より構成される数字以外の文字で始まる文字列で表されます。
シンボルの例を以下に示します。

- _TASK1
- IDLE3

関数名

関数名は数字、英大文字、英小文字、'_'(アンダースコア)、'$'(ドル記号)より構成される数字以外の文字で始まり、'()'で終わる文字列で表されます。
C 言語で記述した関数名の例を以下に示します。

- main()
- func()

アセンブリ言語で記述した場合はモジュールの先頭ラベルを関数名とします。

周波数

周波数は数字と'.'(ピリオド) から構成され'MHz'で終わる文字列で表されます。小数点以下は 6 桁が有効です。なお周波数は 10 進数のみで記述可能です。
周波数の例を以下に示します。

- 16MHz
- 8.1234MHz

なお、周波数は'.'で始まってはいけません。

時間

時間は数字と'. '（ピリオド）から構成され'ms'で終わる文字列で表されます。有効桁数は'ms'の場合小数点以下 3 桁です。なお時間は 10 進数のみで記述可能です。
時間の例を以下に示します。

- 10ms
- 10.5ms

なお時間は'.'(ピリオド)で始まってはいけません。

6.1.2 コンフィギュレーションファイルの定義項目

コンフィギュレーションファイルでは以下の項目[56]の定義をおこないます。

- システム定義
- システムクロック定義
- 最大項目定義
- タスク定義
- イベントフラグ定義
- セマフォ定義
- メールボックス定義
- データキュー定義
- short データキュー定義
- 固定長メモリープール定義
- 可変長メモリープール定義
- 周期ハンドラ定義
- アラームハンドラ定義
- 割り込みベクタ定義

【システム定義】

<< 形式 >>

```
// System Definition
system{
    stack_size      = システムスタックサイズ ;
    priority        = 優先度の最大値 ;
    system_IPL      = カーネルマスクレベル(OS割り込み禁止レベル) ;
    timeout         = タイムアウト ;
    task_pause      = タスクポーズ ;
    tic_deno        = タイムティック分母 ;
    tic_nume        = タイムティック分子 ;
    message_pri     = 最大メッセージ優先度値;
};
```

<< 内容 >>

1. **システムスタックサイズ (バイト)**

   【 定義形式 】数値

   【 定義範囲 】6 以上

   【 デフォルト値 】400H

   サービスコール処理および割り込み処理で使用するスタックサイズの合計を定義します。

[56] タスク定義以外の項目は、省略することができます。省略した場合にはデフォルトコンフィギュレーションファイルの定義が参照されます。

2. 優先度の最大値（最低優先度の値）

【 定義形式 】数値

【 定義範囲 】1 ～ 255

【 デフォルト値 】63

MR308 のアプリケーションプログラムの使用する優先度の最大値を定義します。すなわち使用している優先度の最も大きい値を設定してください。[57]

3. カーネルマスクレベル(OS 割り込み禁止レベル)

【 定義形式 】数値

【 定義範囲 】1 ～ 7

【 デフォルト値 】7

サービスコール内での IPL の値、すなわちカーネルマスクレベル(OS 割り込み禁止レベル)を設定します。

4. タイムアウト

【 定義形式 】シンボル

【 定義範囲 】YES or NO

【 デフォルト値 】NO

tslp_tsk、twai_flg、twai_sem、trcv_mbx、tsnd_dtq、trcv_dtq、tget_mpf、vtsnd_dtq、vtrcv_dtq を使用している場合は、YES を設定し、使用していない場合は、NO を設定してください。

5. タスクポーズ

【 定義形式 】シンボル

【 定義範囲 】YES or NO

【 デフォルト値 】NO

デバッガの OS デバッグ機能であるタスクポーズ機能をご使用になる場合は、YES を設定し、使用していない場合は、NO を設定してください。

6. タイムティック分母

【 定義形式 】数値

【 定義範囲 】1 固定

【 デフォルト値 】1

タイムティックの分母を設定します。

[57] MR308 の優先度は、値が大きいほど優先度は低くなります。

7. タイムティック分子

【 定義形式 】数値

【 定義範囲 】1～65535

【 デフォルト値 】1

タイムティックの分子を設定します。タイムティック分母、分子の設定によってシステムクロックの割り込み間隔が決定されます。間隔は、(タイムティック分子/タイムティック分母)ms となります。すなわち、タイムティック分子 ms となります。
マイコンの仕様により、M32C/82,83 を使用し、20MHz 動作時に指定できる tic_nume の値は 26ms となります。

8. 最大メッセージ優先度値

【 定義形式 】数値

【 定義範囲 】1 ～ 255

【 デフォルト値 】なし

メッセージ優先度の最大値を定義します。

【システムクロック定義】

<< 形式 >>

```
// System Clock Definition
clock{
    timer_clock     = MPUのクロック;
    timer           = システムクロックに使用するタイマ;
    IPL             = システムクロック割り込み優先レベル;
};
```

<< 内容 >>

1. MPU のクロック（MHz）

【 定義形式 】周波数

【 定義範囲 】なし

【 デフォルト値 】20MHz

マイコンの MPU 動作クロックの周波数を MHz 単位で定義します。
M16C/70 シリーズの場合は、周辺機能動作クロック f_1 に設定する値と同じ値を設定します。M16C/70 シリーズの場合、コンパイルオプションの設定が必要になります。詳細は- 94 -ページを参照ください。

2. システムクロックに使用するタイマ

【 定義形式 】シンボル

【 定義範囲 】A0, A1, A2, A3, A4, B0, B1, B2, B3, B4, B5, OTHER, NOTIMER

【 デフォルト値 】NOTIMER

システムクロックに使用するハードウェアタイマを定義します。
システムクロックを使用しない場合は、"NOTIMER"を定義します。

3. システムクロック割り込み優先レベル

【 定義形式 】数値

【 定義範囲 】1 ～ (システム定義のカーネルマスクレベル(OS 割り込み禁止レベル))

【 デフォルト値 】4

システムクロック用タイマ割り込みの優先レベルを定義します。1 ～ カーネルマスクレベル(OS 割り込み禁止レベル)までの値を設定して下さい。
システムクロックの割り込みハンドラ処理中は、ここで定義した割り込みレベルより低いレベルの割り込みは受け付けられません。

【最大項目数定義】

この定義は、別 ROM 化[58]を行う場合のみ設定する項目です。
ここでの設定は、複数のアプリケーションの中で、各定義の最大数を定義します。

<< 形式 >>

```
// Max Definition
maxdefine{
    max_task        = 最大タスク定義数;
    max_flag        = 最大イベントフラグ定義数;
    max_dtq         = 最大データキュー定義数;
    max_mbx         = 最大メールボックス定義数;
    max_sem         = 最大セマフォ定義数;
    max_mpf         = 最大固定長メモリプール定義数;
    max_mpl         = 最大可変長メモリプール定義数;
    max_cyh         = 最大周期ハンドラ定義数;
    max_alh         = 最大アラームハンドラ定義数;
    max_vdtq        = 最大shortデータキュー定義数;
};
```

<< 内容 >>

1. 最大タスク定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 255

【 デフォルト値 】なし

タスク定義の最大数を定義します。

2. 最大イベントフラグ定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 255

【 デフォルト値 】なし

イベントフラグ定義の最大数を定義します。

[58] 別 ROM 化の詳細については、- 146 -ページを参照してください。

3. 最大データキュー定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 255

【 デフォルト値 】なし

データキュー定義の最大数を定義します。

4. 最大メールボックス定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 255

【 デフォルト値 】なし

メールボックス定義の最大数を定義します。

5. 最大セマフォ定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 255

【 デフォルト値 】なし

セマフォ定義の最大数を定義します。

6. 最大固定長メモリプール定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 255

【 デフォルト値 】なし

固定長メモリプール定義の最大数を定義します。

7. 最大可変長メモリプール定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 255

【 デフォルト値 】なし

可変長メモリプール定義の最大数を定義します。

8. 最大周期ハンドラ定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 255

【 デフォルト値 】なし

周期ハンドラ定義の最大数を定義します。

9. 最大アラームハンドラ定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 255

【 デフォルト値 】なし

アラームハンドラ定義の最大数を定義します。

10. 最大 short データキュー定義数

【 定義形式 】数値

【 定義範囲 】1 ～ 255

【 デフォルト値 】なし

short データキュー定義の最大数を定義します。

【タスク定義】

<< 形式 >>

```
 // Tasks_Definition
task[ID番号]{
    name                    = ID名称;
    entry_address           = タスクの開始アドレス;
    stack_size              = タスクのユーザースタックサイズ;
    priority                = タスクの初期優先度;
    context                 = 使用するレジスタ;
    stack_section           = スタックを配置するセクション名;
    initial_start           = TA_ACT属性(初期起動状態);
    exinf                   = 拡張情報;
};
            :
            :
```

ID 番号は 1～255 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>
タスク ID 番号ごとに以下の定義をおこないます。

1. タスク ID 名称

【 定義形式 】シンボル

【 定義範囲 】なし

【 デフォルト値 】なし

タスクの ID 名称を定義します。なお、ここで定義した関数名は、以下のように kernel_id.h ファイルに出力されます。

```
#define タスクID名称 タスクID
```

2. タスク開始アドレス

【 定義形式 】シンボル、または、関数名

【 定義範囲 】なし

【 デフォルト値 】なし

タスクの入り口アドレスを定義します。C 言語で記述したときはその関数名の最後に ()をつけるか、先頭に_をつけます。
なお、ここで定義した関数名は、kernel_id.h ファイルに以下の宣言文が出力されます。

```
#pragma TASK /V4 関数名
```

3. ユーザースタックサイズ (バイト)

【 定義形式 】数値

【 定義範囲 】12 以上

【 デフォルト値 】256

タスクごとのユーザースタックサイズを定義します。ユーザースタックとは、各々のタスクが使用するスタック領域です。MR308 ではユーザー用のスタック領域をタスクごとに割り当てる必要があり最低で 12 バイトが必要です。

4. タスクの初期優先度

【 定義形式 】数値

【 定義範囲 】1 ～ (システム定義の優先度の最大値)

【 デフォルト値 】1

タスクの起動時の優先度を定義します。
MR308 の優先度は、値が小さいほど、優先度としては高くなります。

5. 使用するレジスタ

【 定義形式 】シンボル [,シンボル,.....]

【 定義範囲 】R0,R1,R2,R3,A0,A1,SB,FB から選択

【 デフォルト値 】全レジスタ

タスクで使用するレジスタを定義します。MR308 では、ここで定義されたレジスタをコンテキストとして扱います。 タスク起動時に、タスク起動コードが R0 レジスタに設定されますので、R0 レジスタは、必ず指定して下さい。ただし、タスクをアセンブリ言語で記述する場合のみ使用レジスタが選択可能です。C 言語で記述する場合、全レジスタを選択してください。
なお、レジスタを選択する場合、各タスクで使用しているサービスコールのパラメータを格納するレジスタについては、全て選択してください。
MR308 カーネル内では、レジスタバンク切り替えは行いません。
本定義を省略した場合、全レジスタが選択されたものとします。

6. スタックを配置するセクション名

【 定義形式 】シンボル

【 定義範囲 】なし

【 デフォルト値 】stack

スタックを配置するセクション名を定義します。ここで定義したセクションは、必ず、セクションファイル (asm_sec.inc あるいは c_sec.inc)にて配置を行って下さい。
定義しない場合は、stack セクションに配置します。

7. TA_ACT 属性(初期起動状態)

【 定義形式 】シンボル

【 定義範囲 】ON or OFF

【 デフォルト値 】OFF

タスクの初期起動状態を定義します。
ON を指定すると、システムの初期起動時に READY 状態になります。

8. 拡張情報

【 定義形式 】数値

【 定義範囲 】0～0xFFFFFFFF

【 デフォルト値 】0

タスクの拡張情報を定義します。起動要求のキューイングによってタスクが再起動する際などに引数として渡されます。

【イベントフラグ定義】
この定義は、イベントフラグ機能を使用する場合に必ず設定する項目です。

<< 形式 >>

```
// Eventflag Definition
flag[ID番号]{
    name            = ID名称;
    wait_queue      = イベントフラグ待ちキュー選択;
    initial_pattern = イベントフラグ初期値;
    wait_multi      = 複数待ち属性;
    clear_attribute = クリア属性;
};
            :
            :
```

ID 番号は 1～255 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

イベントフラグ ID 番号ごとに以下の定義をおこないます。

1. ID 名称

【 定義形式 】シンボル

【 定義範囲 】なし

【 デフォルト値 】なし

プログラム中でイベントフラグを指定する時の名前を定義します。

2. イベントフラグ待ちキュー選択

【 定義形式 】シンボル

【 定義範囲 】TA_TFIFO,TA_TPRI

【 デフォルト値 】TA_TFIFO

イベントフラグ待ちの方法を選択します。TA_TFIFO は、FIFO 順で待ちキューにつながれます。TA_TPRI は、タスクの優先度の高い順に待ちキューにつながれます。

3. イベントフラグ初期値

【 定義形式 】数値

【 定義範囲 】0～0xFFFF

【 デフォルト値 】0

イベントフラグの初期ビットパターンを指定します。

4. 複数待ち属性

【 定義形式 】シンボル

【 定義範囲 】TA_WMUL,TA_WSGL

【 デフォルト値 】TA_WSGL

イベントフラグ待ちキューに複数のタスクがつなぐことを許すかどうか指定します。TA_WMUL の場合、TA_WMUL 属性が付加され、複数タスクの待ちを許します。TA_WSGL の場合、TA_WSGL 属性が付加され、複数タスクの待ちを許しません。

5. クリア属性

【 定義形式 】シンボル

【 定義範囲 】YES,NO

【 デフォルト値 】NO

イベントフラグ属性として、TA_CLR 属性を付加するかどうかを指定します。YES の場合、TA_CLR 属性が付加されます。NO の場合、TA_CLR 属性は付加されません。

【セマフォ定義】
この定義は、セマフォ機能を使用する場合に必ず設定する項目です。

<< 形式 >>

```
// Semaphore Definition
semaphore[ID番号]{
    name                    = ID名称;
    wait_queue              = セマフォ待ちキューの選択;
    initial_count           = セマフォのカウンタ初期値;
    max_count               = セマフォのカウンタの最大値;
};
            :
            :
```

ID 番号は 1～255 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

セマフォ ID 番号ごとに以下の定義をおこないます。

1. **ID 名称**

   【 定義形式 】シンボル

   【 定義範囲 】なし

   【 デフォルト値 】なし

   プログラム中でセマフォを指定する時の名前を定義します。

2. **セマフォ待ちキューの選択**

   【 定義形式 】シンボル

   【 定義範囲 】TA_TFIFO,TA_TPRI

   【 デフォルト値 】TA_TFIFO

   セマフォ待ちの方法を選択します。TA_TFIFO は、FIFO 順で待ちキューにつながれます。TA_TPRI は、タスクの優先度の高い順に待ちキューにつながれます。

3. **セマフォカウンタ初期値**

   【 定義形式 】数値

   【 定義範囲 】0 ～ 65535

   【 デフォルト値 】1

   セマフォカウンタの初期値を定義します。

4. セマフォカウンタ最大値

【 定義形式 】数値

【 定義範囲 】1 ～ 65535

【 デフォルト値 】1

セマフォカウンタの最大値を定義します。

【データキュー定義】
この定義は、データキュー機能を使用する場合に必ず設定する項目です。

<< 形式 >>

```
// Dataqueue Definition
dataqueue[ID番号]{
    name            = ID名称;
    buffer_size     = データキュー個数;
    wait_queue      = データキュー待ちキュー選択;
};
            :
            :
```

ID 番号は 1～255 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

データキューID 番号ごとに以下の項目の定義をおこないます。

5. ID 名称

【 定義形式 】シンボル

【 定義範囲 】なし

【 デフォルト値 】なし

プログラム中でデータキューを指定する時の名前を定義します。

6. データ個数

【 定義形式 】数値

【 定義範囲 】0～0x1FFF

【 デフォルト値 】0

送信可能なデータの個数を指定します。指定するのはサイズではなく個数です。

7. データキュー待ちキューの選択

【 定義形式 】シンボル

【 定義範囲 】TA_TFIFO,TA_TPRI

【 デフォルト値 】TA_TFIFO

データキュー送信待ちの方法を選択します。TA_TFIFO は、FIFO 順で待ちキューにつながれます。

TA_TPRI は、タスクの優先度の高い順に待ちキューにつながれます。

【short データキュー定義】
この定義は、short データキュー機能を使用する場合に必ず設定する項目です。

<< 形式 >>

```
// Vdataqueue Definition
vdataqueue[ID番号]{
    name            = ID名称;
    buffer_size     = データキュー個数;
    wait_queue      = データキュー待ちキュー選択;
};
            :
            :
```

ID 番号は 1～255 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

short データキューID 番号ごとに以下の項目の定義をおこないます。

1. **ID 名称**

   【 定義形式 】シンボル

   【 定義範囲 】なし

   【 デフォルト値 】なし

   プログラム中で short データキューを指定する時の名前を定義します。

2. **データ個数**

   【 定義形式 】数値

   【 定義範囲 】0～0x3FFF

   【 デフォルト値 】0

   送信可能なデータの個数を指定します。指定するのはサイズではなく個数です。

3. **short データキュー待ちキューの選択**

   【 定義形式 】シンボル

   【 定義範囲 】TA_TFIFO,TA_TPRI

   【 デフォルト値 】TA_TFIFO

   short データキュー送信待ちの方法を選択します。TA_TFIFO は、FIFO 順で待ちキューにつながれます。TA_TPRI は、タスクの優先度の高い順に待ちキューにつながれます。

【メールボックス定義】
この定義は、メールボックス機能を使用する場合に必ず設定する項目です。

<< 形式 >>

```
// Mailbox Definition
mailbox[ID番号]{
    name            = ID名称;
    wait_queue      = メールボックス待ちキュー選択;
    message_queue   = メッセージキュー選択;
    max_pri         = メッセージ最大優先度;
};
            :
            :
```

ID 番号は 1～255 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

メールボックス ID 番号ごとに以下の項目の定義をおこないます。

1. ID 名称

【 定義形式 】シンボル

【 定義範囲 】なし

【 デフォルト値 】なし

プログラム中でメールボックスを指定する時の名前を定義します。

2. メールボックス待ちキューの選択

【 定義形式 】シンボル

【 定義範囲 】TA_TFIFO,TA_TPRI

【 デフォルト値 】TA_TFIFO

メールボックス待ちの方法を選択します。TA_TFIFO は、FIFO 順で待ちキューにつながれます。TA_TPRI は、タスクの優先度の高い順に待ちキューにつながれます。

3. メッセージキュー選択

【 定義形式 】シンボル

【 定義範囲 】TA_MFIFO,TA_MPRI

【 デフォルト値 】TA_MFIFO

メールボックスのメッセージキュー選択の方法を選択します。TA_MFIFO は、FIFO 順でキューにつながれます。TA_MPRI は、メッセージの優先度の高い順に待ちキューにつながれます。

4. メッセージキュー最大優先度

【 定義形式 】数値

【 定義範囲 】1～「最大項目数定義」の「メッセージ優先度最大値」で指定した値。

【 デフォルト値 】1

メールボックスのメッセージの最大優先度を指定します。

【固定長メモリプール定義】
この定義は、固定長メモリプール機能を使用する場合に必ず設定する項目です。

<< 形式 >>

```
// Fixed Memorypool Definition
memorypool[ID番号]{
    name            = ID名称;
    section         = セクション名;
    num_block       = メモリプールのブロック数;
    siz_block       = メモリプールのブロックサイズ;
    wait_queue      = メモリプール待ちキュー選択;
};
```

ID 番号は、1～255 の範囲でなければなりません。ID 番号は、省略可能です。
省略した場合は番号を小さい方から順に自動的に割り当てます。

<< 内容 >>

メモリプール ID 番号ごとに以下の定義をおこないます。

1. ID 名称

【 定義形式 】シンボル

【 定義範囲 】なし

【 デフォルト値 】なし

プログラム中でメモリプールを指定する時の名前を指定します。

2. セクション名

【 定義形式 】シンボル

【 定義範囲 】なし

【 デフォルト値 】MR_HEAP

メモリプールを配置するセクションの名前を定義します。ここで定義したセクションは、必ず、セクションファイル (asm_sec.inc あるいは c_sec.inc)にて配置を行って下さい。
定義しない場合は、MR_HEAP セクションに配置します。

3. ブロック数

【 定義形式 】数値

【 定義範囲 】1～65535

【 デフォルト値 】1

メモリプールのブロック総数を定義します。

4. サイズ（バイト）

【 定義形式 】数値

【 定義範囲 】4～65535

【 デフォルト値 】256

メモリプールの 1 ブロック当たりのサイズを定義します。この定義によりメモリプールとして使用する RAM 容量は、 (ブロック数)×(サイズ)バイトです。

5. メモリプール待ちキューの選択

【 定義形式 】シンボル

【 定義範囲 】TA_TFIFO,TA_TPRI

【 デフォルト値 】TA_TFIFO

固定長メモリプール獲得待ちの方法を選択します。TA_TFIFO は、FIFO 順で待ちキューにつながれます。TA_TPRI は、タスクの優先度の高い順に待ちキューにつながれます。

【可変長メモリプール定義】

この定義は、可変長メモリプール機能を使用する場合に必ず設定する項目です。

<< 形式 >>

```
// Variable-Size Memorypool Definition
variable_memorypool[ID番号] {
    name            = ID名称;
    max_memsize     = 確保するメモリブロックサイズの最大値;
    mpl_section     = セクション名;
    heap_size       = メモリプールのサイズ;
};
```

ID 番号は、1～255 の範囲でなければなりません。ID 番号は、省略可能です。
省略した場合は番号を小さい方から順に自動的に割り当てます。

<< 内容 >>

1. ID 名称

【 定義形式 】シンボル

【 定義範囲 】なし

【 デフォルト値 】なし

プログラム中でメモリプールを指定する時の名前を指定します。

2. 確保するメモリブロックサイズの最大値（バイト）

【 定義形式 】数値

【 定義範囲 】1 ～ 65520

【 デフォルト値 】なし

アプリケーションプログラム中で、確保しているメモリブロックサイズの最大値を指定します。

3. セクション名

【 定義形式 】シンボル

【 定義範囲 】なし

【 デフォルト値 】MR_HEAP

メモリプールを配置するセクションの名前を定義します。ここで定義したセクションは、必ず、セクションファイル（asm_sec.inc あるいは c_sec.inc)にて配置を行って下さい。
定義しない場合は、MR_HEAP セクションに配置します。

4. メモリプールのサイズ（バイト）

【 定義形式 】数値

【 定義範囲 】16 ～ 0xFFFFFF

【 デフォルト値 】なし

メモリプールのサイズを指定します。
MR308 では、メモリを 4 種類の固定長ブロックサイズで管理しています。この 4 種類のブロックサイズ(下記 a,b,c,d)は、下記の計算式から算出されます。

a = (((max_memsize+(X-1)/(X×8))+1) ×X
b = a × 2
c = a × 4
d = a × 8
X: ブロック管理用データサイズ(1 ブロックあたり 8 バイト)

可変長メモリプールは、メモリブロックを管理する領域として 1 ブロックあたり 8 バイトの領域が必要となります。このため、max_memsize で指定したサイズのメモリブロックを獲得するためには、**max_memsize+8** の結果が収まる上記 a,b,c,d いずれかのブロックサイズ以上の値をメモリプールのサイズに指定しなければなりません。

【周期ハンドラ定義】
この定義は、周期ハンドラ機能を使用する場合に必ず設定する項目です。

<< 形式 >>

```
// Cyclic Handlar Definition
 cyclic_hand[ID番号]{
    name                    = ID名称;
    interval_counter        = 周期ハンドラの周期間隔;
    start                   = TA_STA属性;
    phsatr                  = TA_PHS属性;
    phs_counter             = 起動位相;
    entry_address           = 周期ハンドラの開始アドレス;
    exinf                   = 拡張情報;
 };
            :
            :
```

ID 番号は 1 ～ 255 の範囲でなければなりません。ID 番号は省略可能です。
省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

周期ハンドラ ID 番号ごとに以下の項目の定義をおこないます。

1. **ID 名称**

   【 定義形式 】シンボル

   【 定義範囲 】なし

   【 デフォルト値 】なし

   プログラム中で周期ハンドラを指定する時の名前を指定します。

2. **周期間隔**

   【 定義形式 】数値

   【 定義範囲 】1 ～ 0x7FFFFFFF

   【 デフォルト値 】なし

   周期ハンドラの周期起動間隔を定義します。ここで定義する時間の単位は ms です。例えば、1 秒間隔で周期起動しようとすると、この値を 1000 に設定します。

3. **TA_STA 属性**

   【 定義形式 】シンボル

   【 定義範囲 】ON,OFF

   【 デフォルト値 】OFF

   周期ハンドラの TA_STA 属性を指定します。ON の場合は、TA_STA 属性が付加され、OFF の場合は、TA_STA 属性は付加されません。

4. TA_PHS 属性

【 定義形式 】シンボル

【 定義範囲 】ON,OFF

【 デフォルト値 】OFF

周期ハンドラの TA_PHS 属性を指定します。ON の場合は、TA_PHS 属性が付加され、OFF の場合は、TA_PHS 属性は付加されません。

5. 起動位相

【 定義形式 】数値

【 定義範囲 】1 ～ 0x7FFFFFFF

【 デフォルト値 】なし

周期ハンドラの起動位相を定義します。ここで定義する時間の単位は ms です。

6. 開始アドレス

【 定義形式 】シンボル、または、関数名

【 定義範囲 】なし

【 デフォルト値 】なし

周期ハンドラの開始アドレスを定義します。
なお、ここで定義した関数名は、kernel_id.h ファイルに以下の宣言文が出力されます。

```
#pragma CYCHANDLER 関数名
```

7. 拡張情報

【 定義形式 】数値

【 定義範囲 】0～0xFFFFFFFF

【 デフォルト値 】0

周期ハンドラの拡張情報を定義します。周期ハンドラを起動する際に引数として渡されます。

【アラームハンドラ定義】
この定義は、アラームハンドラ機能を使用する場合に必ず設定する項目です。

<< 形式 >>

```
// Alarm Handlar Definition
alarm_hand[ID番号]{
    name                    = ID名称;
    entry_address           = アラームハンドラの開始アドレス;
    exinf                   = 拡張情報;
};
            :
            :
```

ID 番号は 1～255 の範囲でなければなりません。ID 番号は省略可能です。

省略した場合は番号を小さいほうから順に自動的に割り当てます。

<< 内容 >>

アラームハンドラ ID 番号ごとに以下の項目の定義をおこないます。

1. **ID 名称**

   【 定義形式 】シンボル

   【 定義範囲 】なし

   【 デフォルト値 】なし

   プログラム中でアラームハンドラを指定する時の名前を指定します。

2. **開始アドレス**

   【 定義形式 】シンボル、または、関数名

   【 定義範囲 】なし

   アラームハンドラの開始アドレスを定義します。なお、ここで定義した関数名は、kernel_id.h ファイルに以下の宣言文が出力されます。

   ```
   #pragma ALMHANDLER 関数名
   ```

3. **拡張情報**

   【 定義形式 】数値

   【 定義範囲 】0～0xFFFFFFFF

   【 デフォルト値 】0

   アラームハンドラの拡張情報を定義します。アラームハンドラを起動する際に引数として渡されます。

【割り込みベクタ定義】

この定義は、割り込みハンドラを使用する場合に設定する項目です。

<< 形式 >>

```
    // Interrupt Vector Definition
    interrupt_vector[ベクタ番号] {
        os_int                  = カーネル管理(OS依存)割り込みハンドラ;
        entry_address           = 割り込みハンドラの開始アドレス;
pragma_switch ~~pragma_swicth~~ = PRAGMA拡張機能に渡すスイッチ;
    };
                :
                :
```

割り込みベクタ番号は 0～63、247～255 の範囲まで記述できます。
ただし、そのベクタ番号が有効か否かは使用しているマイクロコンピュータに依存します。
M16C/80 シリーズの場合の割り込み要因と割り込みベクタ番号は表 6.3に示す対応になります。
また、コンフィギュレータは、ここで指定した割り込みの割り込み制御レジスタ（IPL 等）や、割り込み要因等の初期設定のコードは生成しません。初期設定はスタートアップファイル中もしくは、開発されるアプリケーションにあわせて作成して頂く必要があります。

<< 内容 >>

1. カーネル管理(OS 依存)割り込みハンドラ

【 定義形式 】シンボル

【 定義範囲 】YES または NO

【 デフォルト値 】なし

ハンドラが カーネル管理(OS 依存)割り込みハンドラかどうかを定義します。カーネル管理(OS 依存)割り込みハンドラであれば YES を、カーネル管理外(OS 独立)割り込みハンドラであれば NO を定義して下さい。

YES を定義した場合、kernel_id.h ファイルに以下の宣言文を出力します。

```
#pragma INTHANDLER /V4 関数名
```

また、NO を定義した場合、kernel_id.h ファイルに以下の宣言文を出力します。

```
#pragma INTERRUPT /V4 関数名
```

2. 開始アドレス

【 定義形式 】シンボルまたは関数名

【 定義範囲 】なし

【 デフォルト値 】__SYS_DMY_INH

割り込みハンドラの入口アドレスを定義します。C 言語で記述した時はその関数名の最後に () をつけるか先頭に_をつけます。

3. PRAGMA 拡張機能に渡すスイッチ

【 定義形式 】シンボル

【 定義範囲 】E,F,B

【 デフォルト値 】なし

#pragma INTHANDLER や、#pragma INTERRUPT に渡すスイッチを指定します。”E”を指定した場合、”/E”スイッチが指定され、多重割り込みが許可されます。”F”を指定した場合、”/F”スイッチが指定され、割り込みハンドラからのリターン時に”FREIT”命令が出力されます。”B”を指定した場合は、”/B”スイッチが指定され、レジスタバンク 1 が指定されます。

複数のスイッチを同時に指定することも出来ます。ただし、カーネル管理(OS 依存)割り込みハンドラの場合は、”E”スイッチのみ指定することが出来ます。カーネル管理外(OS 独立)割り込みハンドラの場合は、”E,F,B”スイッチを指定することが出来ますが、”E”と”B”を同時に指定することは出来ません。

注意事項

1．レジスタバンク指定方法について

C 言語でレジスタバンク 1 のレジスタを使ったカーネル管理(OS 依存)割り込みハンドラの記述はできません。 アセンブリ言語のみ記述することができます。 アセンブリ言語で記述する場合、割り込みハンドラの入口と出口を以下に示すように記述してください。

(ret_int サービスコールを発行する前に必ず B フラグをクリアしてください。)

例）interrupt:

```
        fset    B
                :
        fclr    B
        ret_int
```

MR308 カーネル内では、レジスタバンク切り替えは行いません。

２．高速割り込み指定方法について

高速割り込みを有効に使用する為に、高速割り込み内では、レジスタバンク 1 のレジスタを使用してください。 また、高速割り込みは、カーネル管理(OS 依存)割り込みハンドラには、使用できません。

３．NMI 割り込み、監視タイマ割り込みは、カーネル管理(OS 依存)割り込みで使用しないでください。

表 6.3　M16C/80 シリーズでの割り込み要因とベクタ番号との対応

割り込み要因	割り込みベクタ番号	セクション名
DMA0	8	INTERRUPT_VECTOR
DMA1	9	INTERRUPT_VECTOR
DMA2	10	INTERRUPT_VECTOR
DMA3	11	INTERRUPT_VECTOR
タイマ A0	12	INTERRUPT_VECTOR
タイマ A1	13	INTERRUPT_VECTOR
タイマ A2	14	INTERRUPT_VECTOR
タイマ A3	15	INTERRUPT_VECTOR
タイマ A4	16	INTERRUPT_VECTOR
UART0 送信	17	INTERRUPT_VECTOR
UART0 受信	18	INTERRUPT_VECTOR
UART1 送信	19	INTERRUPT_VECTOR
UART1 受信	20	INTERRUPT_VECTOR
タイマ B0	21	INTERRUPT_VECTOR
タイマ B1	22	INTERRUPT_VECTOR
タイマ B2	23	INTERRUPT_VECTOR
タイマ B3	24	INTERRUPT_VECTOR
タイマ B4	25	INTERRUPT_VECTOR
INT5 外部割り込み	26	INTERRUPT_VECTOR
INT4 外部割り込み	27	INTERRUPT_VECTOR
INT3 外部割り込み	28	INTERRUPT_VECTOR
INT2 外部割り込み	29	INTERRUPT_VECTOR
INT1 外部割り込み	30	INTERRUPT_VECTOR
INT0 外部割り込み	31	INTERRUPT_VECTOR
タイマ B5	32	INTERRUPT_VECTOR
UART2 送信/NACK	33	INTERRUPT_VECTOR
UART2 受信/ACK	34	INTERRUPT_VECTOR
UART3 送信/NACK	35	INTERRUPT_VECTOR
UART3 受信/ACK	36	INTERRUPT_VECTOR
UART4 送信/NACK	37	INTERRUPT_VECTOR
UART4 受信/ACK	38	INTERRUPT_VECTOR
バス衝突検出 (UART2)	39	INTERRUPT_VECTOR
バス衝突検出 (UART3)	40	INTERRUPT_VECTOR
バス衝突検出 (UART4)	41	INTERRUPT_VECTOR
A/D	42	INTERRUPT_VECTOR
キー入力割り込み	43	INTERRUPT_VECTOR
ユーザーソフトウエア割り込み	44	INTERRUPT_VECTOR
:		INTERRUPT_VECTOR
ユーザーソフトウエア割り込み	54	INTERRUPT_VECTOR
MR308 用ソフトウエア割り込み	55	INTERRUPT_VECTOR
ユーザーソフトウエア割り込み	56	INTERRUPT_VECTOR
ユーザーソフトウエア割り込み	57	INTERRUPT_VECTOR
MR308 用ソフトウエア割り込み	58	INTERRUPT_VECTOR
:		INTERRUPT_VECTOR
MR308 用ソフトウエア割り込み	63	INTERRUPT_VECTOR
未定義命令	247	FIX_INTERRUPT_VECTOR
オーバーフロー	248	FIX_INTERRUPT_VECTOR
BRK 命令	249	FIX_INTERRUPT_VECTOR
アドレス一致	250	FIX_INTERRUPT_VECTOR
		FIX_INTERRUPT_VECTOR
監視タイマ	252	FIX_INTERRUPT_VECTOR
		FIX_INTERRUPT_VECTOR
NMI	254	FIX_INTERRUPT_VECTOR
リセット	255	FIX_INTERRUPT_VECTOR

6.1.3 コンフィギュレーションファイル例

以下にコンフィギュレーションファイルの例を示します。

```
 1 ////////////////////////////////////////////////////////////////////////////
 2 //
 3 //    kernel.cfg : building file for MR308 Ver.4.00
 4 //
 5 //    Generated by M3T-MR308 GUI Configurator at 2005/02/28 19:01:20
 6 //
 7 ////////////////////////////////////////////////////////////////////////////
 8
 9 // system definition
10 system{
11      stack_size      = 256;
12      system_IPL       = 4;
13      message_pri      = 64;
14      timeout  = NO;
15      task_pause       = NO;
16      tick_nume        = 10;
17      tick_deno        = 1;
18 };
19
20 // max definition
21 maxdefine{
22      max_task        = 3;
23      max_flag        = 4;
24      max_sem = 3;
25      max_dtq = 3;
26      max_mbx = 4;
27      max_mpf = 3;
28      max_mpl = 3;
29      max_cyh = 4;
30      max_alh = 2;
31 };
32
33 // system clock definition
34 clock{
35      timer_clock     = 20.000000MHz;
36      timer   = A0;
37      IPL     = 3;
38 };
39
40 task[]{
41      entry_address   = task1();
42      name    = ID_task1;
43      stack_size      = 256;
44      priority        = 1;
45      initial_start   = OFF;
46      exinf   = 0x0;
47 };
48 task[]{
49      entry_address   = task2();
50      name    = ID_task2;
51      stack_size      = 256;
52      priority        = 5;
53      initial_start   = ON;
54      exinf   = 0xFFFF;
55 };
56 task[3]{
57      entry_address   = task3();
58      name    = ID_task3;
59      stack_size      = 256;
60      priority        = 7;
61      initial_start   = OFF;
62      exinf   = 0x0;
63 };
64
65 flag[]{
66      name    = ID_flg1;
67      initial_pattern = 0x00000000;
68      wait_queue      = TA_TFIFO;
69      clear_attribute = NO;
70      wait_multi      = TA_WSGL;
71 };
72 flag[1]{
73      name    = ID_flg2;
74      initial_pattern = 0x00000001;
```

```
 75      wait_queue      = TA_TFIFO;
 76      clear_attribute = NO;
 77      wait_multi      = TA_WMUL;
 78 };
 79 flag[2]{
 80      name    = ID_flg3;
 81      initial_pattern = 0x0000ffff;
 82      wait_queue      = TA_TPRI;
 83      clear_attribute = YES;
 84      wait_multi      = TA_WMUL;
 85 };
 86 flag[]{
 87      name    = ID_flg4;
 88      initial_pattern = 0x00000008;
 89      wait_queue      = TA_TPRI;
 90      clear_attribute = YES;
 91      wait_multi      = TA_WSGL;
 92 };
 93
 94 semaphore[]{
 95      name    = ID_sem1;
 96      wait_queue      = TA_TFIFO;
 97      initial_count   = 0;
 98      max_count       = 10;
 99 };
100 semaphore[2]{
101      name    = ID_sem2;
102      wait_queue      = TA_TFIFO;
103      initial_count   = 5;
104      max_count       = 10;
105 };
106 semaphore[]{
107      name    = ID_sem3;
108      wait_queue      = TA_TPRI;
109      initial_count   = 255;
110      max_count       = 255;
111 };
112
113 dataqueue[]{
114      name    = ID_dtq1;
115      wait_queue      = TA_TFIFO;
116      buffer_size     = 10;
117 };
118 dataqueue[2]{
119      name    = ID_dtq2;
120      wait_queue      = TA_TPRI;
121      buffer_size     = 5;
122 };
123 dataqueue[3]{
124      name    = ID_dtq3;
125      wait_queue      = TA_TFIFO;
126      buffer_size     = 256;
127 };
128
129 mailbox[]{
130      name    = ID_mbx1;
131      wait_queue      = TA_TFIFO;
132      message_queue   = TA_MFIFO;
133      max_pri = 4;
134 };
135 mailbox[]{
136      name    = ID_mbx2;
137      wait_queue      = TA_TPRI;
138      message_queue   = TA_MPRI;
139      max_pri = 64;
140 };
141 mailbox[]{
142      name    = ID_mbx3;
143      wait_queue      = TA_TFIFO;
144      message_queue   = TA_MPRI;
145      max_pri = 5;
146 };
147 mailbox[4]{
148      name    = ID_mbx4;
149      wait_queue      = TA_TPRI;
150      message_queue   = TA_MFIFO;
151      max_pri = 6;
152 };
153
154 memorypool[]{
```

```
155     name    = ID_mpf1;
156     wait_queue      = TA_TFIFO;
157     section = MR_RAM;
158     siz_block       = 16;
159     num_block       = 5;
160 };
161 memorypool[2]{
162     name    = ID_mpf2;
163     wait_queue      = TA_TPRI;
164     section = MR_RAM;
165     siz_block       = 32;
166     num_block       = 4;
167 };
168 memorypool[3]{
169     name    = ID_mpf3;
170     wait_queue      = TA_TFIFO;
171     section = MPF3;
172     siz_block       = 64;
173     num_block       = 256;
174 };
175
176 variable_memorypool[]{
177     name    = ID_mpl1;
178     max_memsize     = 8;
179     heap_size       = 16;
180 };
181 variable_memorypool[]{
182     name    = ID_mpl2;
183     max_memsize     = 64;
184     heap_size       = 256;
185 };
186 variable_memorypool[3]{
187     name    = ID_mpl3;
188     max_memsize     = 256;
189     heap_size       = 1024;
190 };
191
192 cyclic_hand[]{
193     entry_address   = cyh1();
194     name    = ID_cyh1;
195     exinf   = 0x0;
196     start   = ON;
197     phsatr  = OFF;
198     interval_counter        = 0x1;
199     phs_counter     = 0x0;
200 };
201 cyclic_hand[]{
202     entry_address   = cyh2();
203     name    = ID_cyh2;
204     exinf   = 0x1234;
205     start   = OFF;
206     phsatr  = ON;
207     interval_counter        = 0x20;
208     phs_counter     = 0x10;
209 };
210 cyclic_hand[]{
211     entry_address   = cyh3;
212     name    = ID_cyh3;
213     exinf   = 0xFFFF;
214     start   = ON;
215     phsatr  = OFF;
216     interval_counter        = 0x20;
217     phs_counter     = 0x0;
218 };
219 cyclic_hand[4]{
220     entry_address   = cyh4();
221     name    = ID_cyh4;
222     exinf   = 0x0;
223     start   = ON;
224     phsatr  = ON;
225     interval_counter        = 0x100;
226     phs_counter     = 0x80;
227 };
228
229 alarm_hand[]{
230     entry_address   = alm1();
231     name    = ID_alm1;
232     exinf   = 0xFFFF;
233 };
234 alarm_hand[2]{
```

```
235     entry_address   = alm2;
236     name    = ID_alm2;
237     exinf   = 0x12345678;
238 };
239
240
241 //
242 // End of Configuration
243 //
```

6.2 コンフィギュレータの実行

6.2.1 コンフィギュレータ概要

コンフィギュレータはコンフィギュレーションファイルで定義した内容をアセンブリ言語のインクルードファイル等に変換するツールです。コンフィギュレータの動作概要を図 6.1に示します。
HEW 上でビルドする際は、自動的にコンフィギュレータが起動し、アプリケーションプログラムがビルドされるようになっています。

11. コンフィギュレータの実行には以下の入力ファイルが必要です。

- コンフィギュレーションファイル（XXXX.cfg)
  システムの初期設定項目を記述したファイルです。カレントディレクトリに作成します。

- デフォルトコンフィギュレーションファイル（default.cfg)
  コンフィギュレーションファイルで値の設定を省略した場合にこのファイルに書き込まれている値を設定します環境変数 "LIB308"で示されるディレクトリ、もしくは、カレントディレクトリに置きます。両方のディレクトリに存在する場合は、カレントディレクトリのファイルが優先されます。

- makefile のテンプレートファイル(makefile.dos, makefile, Makefile)
  makefile[59]を生成する場合にテンプレートのファイルとして使用するファイルです。（第 6.2.4 項参照）

- インクルードテンプレートファイル（mr308.inc,sys_ram.inc)
  インクルードファイル mr308.inc,sys_ram.inc のテンプレートとなるファイルです。
  環境変数 "LIB308"で示されるディレクトリに存在します。

- MR308 バージョンファイル（version)
  MR308 のバージョンを記述したファイルです。環境変数 "LIB308" で示されるディレクトリに存在します。コンフィギュレータはこのファイルを読み込み、起動メッセージに MR308 のバージョン情報を出力します。

12. コンフィギュレータの実行によって以下のファイルが出力されます。

コンフィギュレータが出力したファイルには、ユーザーのデータ定義を行わないで下さい。データ定義を行った後で、コンフィギュレータを起動するとユーザーの定義したデータは失われます。

- システムデータ定義ファイル（sys_rom.inc,sys_ram.inc)
  システムの設定を定義しているファイルです。

- インクルードファイル（mr308.inc)
  mr308.inc はアセンブリ言語用のインクルードファイルです。

- システム生成手順記述ファイル（makefile)
  システムを自動生成するためのファイルです。

[59] この makefile は、UNIX 標準もしくは準拠の make コマンドで処理可能なシステム生成手順記述ファイルです。

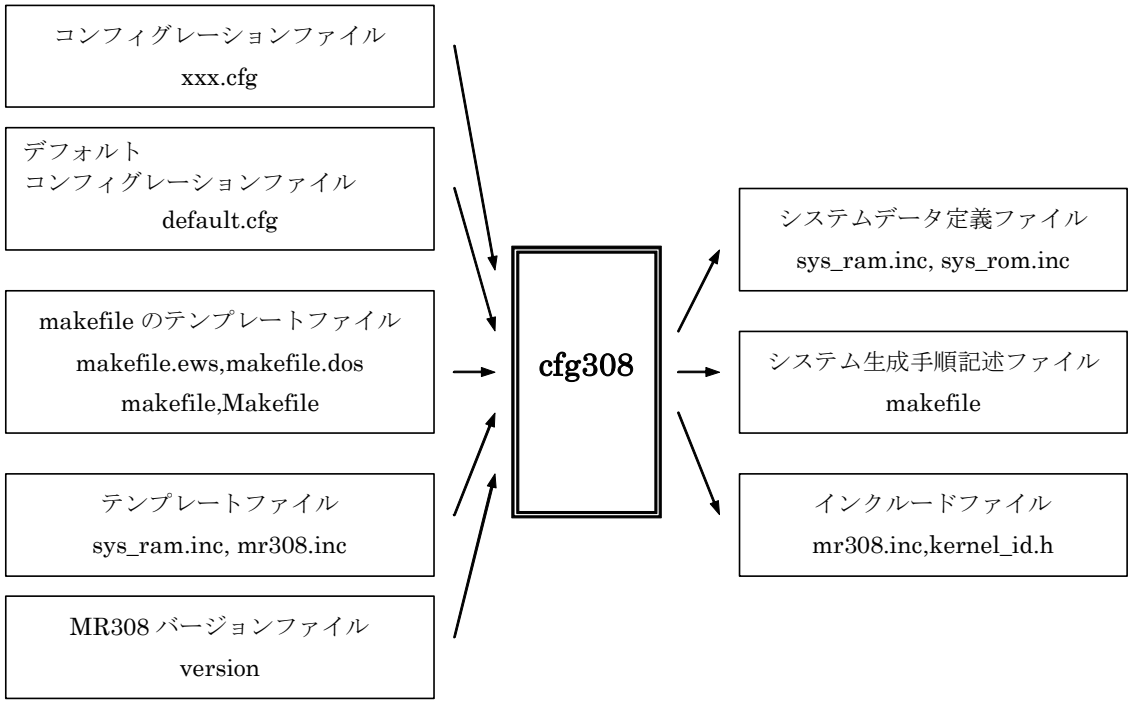


図 6.1　コンフィギュレータ動作概要

6.2.2 コンフィギュレータの環境設定

コンフィギュレータを実行するにあたって環境変数 "LIB308"が正しく設定されているかを確認してください。

環境変数 "LIB308" で示すディレクトリ下には以下のファイルがないと正常に実行できません。

- デフォルトコンフィギュレーションファイル（default.cfg）
  カレントディレクトリにコピーして使用することもできます。その場合はカレントディレクトリのファイルを優先して使用します。

- システム RAM 領域定義データベースファイル（sys_ram.inc）
- mr308.inc のテンプレートファイル（mr308.inc）
- セクション定義ファイル（c_sec.inc または asm_sec.inc）
- スタートアップファイル（crt0mr.a30 または start.a30）
- makefile のテンプレートファイル（makefile.dos）
- MR308 バージョンファイル（version）

6.2.3 コンフィギュレータ起動方法

コンフィギュレータは以下の形式で起動します。

```
A> cfg308  [-vmV]  コンフィギュレーションファイル名
```

コンフィギュレーションファイル名は、通常拡張子（.cfg）かまたは拡張子（.cfg）を除いたファイル名を指定します。

コマンドオプション

-v オプション

コマンドのオプションの説明と詳細なバージョンを表示します。

-V オプション

コマンドが生成するファイルの作成状況を表示します。

-m オプション

UNIX 標準もしくは準拠のシステム生成手順記述ファイル（makefile)を作成します。指定がない場合は makefile を作成しません。[60]

また、カレントディレクトリにスタートアップファイル（crt0mr.a30 または start.a30)とセクション定義ファイル（c_sec.inc、asm_sec.inc）がない場合は、環境変数 LIB308 が示すディレクトリにある、サンプルスタートアップファイルとセクション定義ファイルをカレントディレクトリにコピーします。

[60] UNIX 標準もしくは準拠の”makefile”には、”clean”ターゲットによりワークファイルを削除する機能があります。すなわち、make コマンドで生成されたオブジェクトファイルなどを削除するには以下のように行います。
> make clean

6.2.4 makefile 生成機能

コンフィギュレータは以下の手順で makefile を生成します。

1. ソースファイルの依存関係を調べます。

コンフィギュレータはカレントディレクトリの拡張子が.c と.a30 のファイルをそれぞれ、C 言語とアセンブリ言語として、それらがインクルードするファイルなどの依存関係を調べます。

したがって、ソースファイルを記述する場合には次の 2 点に注意してください。

- ソースファイルはカレントディレクトリに置かなければなりません。
- ソースファイルの拡張子はC言語では'.c'を、アセンブリ言語では'.a30'を使用してください。
- ソースファイルの依存関係には、#if やコメント行に記述して無効化した#include 文に指定したファイルも含みます。

2. makefile へファイルの依存関係を書き込みます。

カレントディレクトリの "makefile"、または"Makefile"または、環境変数"LIB308"で示されるディレクトリの"makefile.dos"を、テンプレートファイルとして、カレントディレクトリに"makefile"を作成します。

6.2.5 コンフィギュレータ実行上の注意

以下にコンフィギュレータ実行上の注意点を示します。

- コンフィギュレータを再度かけ直した場合は、必ず make clean を実行するかオブジェクトファイル(拡張子.r30) を全て消去してから make コマンドを実行してください。リンク時等にエラーが発生する場合があります。
- スタートアッププログラム名、およびセクション定義ファイル名は変更しないでください。変更した場合、コンフィギュレータ実行時にエラーが発生します。
- コンフィギュレータ cfg308 では、UNIX 標準、もしくは準拠の makefile しか生成しません。

6.2.6 コンフィギュレータのエラーと対処方法

以下のメッセージが表示された場合はコンフィギュレータが正常に終了していませんのでコンフィギュレーションファイルを修正の上、再度コンフィギュレータを実行してください。

エラーメッセージ

cfg308 Error : syntax error near line xxx (test.cfg)
コンフィギュレーションファイルに文法エラーがあります。

cfg308 Error : not enough memory
メモリが足りません。

cfg308 Error : illegal option --> <x>
コンフィギュレータのコマンドオプションに誤りがあります。

cfg308 Error : illegal argument --> <xx>
コンフィギュレータの起動形式に誤りがあります。

cfg308 Error : can't write open <XXXX>
XXXX ファイルが作成できません。ディレクトリの属性やディスクの残り容量を確認してください。

cfg308 Error : can't open <XXXX>
XXXX ファイルにアクセスできません。XXXX ファイルの属性や、存在を確認してください。

cfg308 Error : can't open version file
環境変数"LIB308"の示すディレクトリの下に MR308 バージョンファイル"version"がありません。

cfg308 Error : can't open default configuration file
デフォルトコンフィギュレーションファイルがアクセスできません。環境変数 "LIB308"の示すディレクトリ、またはカレントディレクトリに"default.cfg"が必要です。

cfg308 Error : can't open configuration file <xxxxcfg>
コンフィギュレーションファイルがアクセスできません。コンフィギュレータの起動形式を確認の上、正しいファイル名を指定してください。

cfg308 Error : illegal XXXX --> <xx> near line xxx (xxxx.cfg)
定義項目 XXXX の数値または ID 番号が間違っています。定義範囲を確認してください。

cfg308 Error : Unknown XXXX --> <xx> near line xxx (xxxx.cfg)
定義項目 XXXX のシンボル定義が間違っています。定義範囲を確認してください。

cfg308 Error : too big XXXX's ID number --> <xxx> (xxxx.cfg)
XXXX 定義の ID 番号に、定義したオブジェクトの総数を超える値が設定されています。ID 番号がオブジェクトの総数を超えることはありません。

cfg308 Error : too big task[x]'s priority --> <xxx> near line xxx (xxxx.cfg)
ID 番号 x のタスク定義項目の初期優先度が、システム定義項目の優先度値を越えています。

cfg308 Error : too big IPL --> <xxx> near line xxx (xxxx.cfg)
システムクロック定義項目のシステムクロック割り込み優先レベルがシステム定義項目の system IPL 値を越えています。

cfg308 Error : system timer's vector <x>conflict near line xxx
システムクロックの割り込みベクタに、 別の割り込みが定義されています。 割り込みベクタ番号を確認して下さい。

cfg308 Error : XXXX is not defined (xxxx.cfg)
コンフィギュレーションファイルで XXXX の項目の定義が必要です。

cfg308 Error : system's default is not defined
デフォルトコンフィギュレーションファイルで定義が必要な項目です。

cfg308 Error : double definition <XXXX> near line xxx (xxxx.cfg)
項目 XXXX は既に定義されています。確認の上、重複定義を削除してください。

cfg308 Error : double definition XXXX[x] near line xxx (default.cfg)
cfg308 Error : double definition XXXX[x] near line xxx (xxxx.cfg)
項目 XXXX において ID 番号 x は既に登録されています。ID 番号を変更するか重複定義を削除してください。

cfg308 Error : you must define XXXX near line xxx (xxxx.cfg)
XXXX は、省略できない項目です。

cfg308 Error : you must define SYMBOL near line xxx (xxxxcfg)
省略できないシンボルです。

cfg308 Error : start-up-file (XXXX) not found
カレントディレクトリにスタートアップファイル XXXX が見つかりません。スタートアップファイル"start.a30"または "crt0mr.a30"が、カレントディレクトリに必要です。

cfg308 Error : bad start-up-file(XXXX)
カレントディレクトリに不要なスタートアップファイルがあります。

cfg308 Error : no source file
カレントディレクトリにソースファイルがありません。

cfg308 Error : zero divide error near line xxx (xxxx.cfg)
演算式で 0(ゼロ) 除算が発生しました。

cfg308 Error : task[X].stack_size must set XX or more near line xxx (xxxx.cfg)
タスクのスタックサイズを XX バイト以上のサイズをセットしてください。

cfg308 Error : "R0" must exist in task[x].context near line xxxx (xxxx.cfg)
タスクのコンテキスト選択項目では、必ず、R0 レジスタを選択してください。

cfg308 Error : can't define address match interrupt definition for Task Pause Function near line xxxx (xxxx.cfg)
タスクポーズ機能に必要な割り込みベクタに別の割り込みがコンフィギュレーションファイルに定義されています。

cfg308 Error : Set system.timer [system.timeout = YES] near line xxx (xxxx.cfg)
system.timeout = YES の設定にも関わらず、clock 定義において timer 項目が NOTIMER になっています。timer 項目でタイマを設定していください。

警告メッセージ

以下のメッセージは警告ですので、内容が理解できていれば無視してもかまいません。

cfg308 Warning : system is not defined (xxxx.cfg)

cfg308 Warning : system.XXXX is not defined (xxxx.cfg)
コンフィギュレーションファイルでシステム定義またはシステム定義項目 XXXX が省略されています。

cfg308 Warning : system.message_size is not defined (xxxx.cfg)
システム定義中のメッセージサイズ定義項目が省略されています。メールボックス機能のメッセージサイズ (16 または 32)を指定して下さい。

cfg308 Warning : task[x].XXXX is not defined near line xxx (xxxx.cfg)
ID 番号 x のタスク定義項目 XXXX が省略されています。

cfg308 Warning : Already definition XXXX near line xxx (xxxx.cfg)
XXXX は既に定義されています。定義内容は無視されます。確認の上、重複定義を削除してください。

cfg308 Warning : interrupt_vector[x]'s default is not defined (default.cfg)
デフォルトコンフィギュレーションファイルでベクタ番号xの割り込みベクタ定義が抜けています。

cfg308 Warning : interrupt_vector[x]'s default is not defined near line xxx (test.cfg)
コンフィギュレーションファイルのベクタ番号 x の割り込みベクタは、デフォルトコンフィギュレーションファイルに定義されていません。

cfg308 Warning : Initial Start Task not defined
コンフィギュレーションファイルで、初期起動タスクの定義がないためタスク ID 番号 1 のタスクを初期起動タスクとして定義しました。

cfg308 Warning : system.stack_size is an uneven number near line xxx
cfg308 Warning : task[x].stack_size is an uneven number near line xxx
スタックサイズは、偶数サイズを指定してください。

その他のメッセージ

以下のメッセージは makefile を生成する場合にのみ出力される警告メッセージです。コンフィギュレータは要因となった部分を読み飛ばして makefile を生成します。

cfg308 Error : xxxx (line xxx): include format error.
ファイル読み込みの書式が間違っています。正しい書式に書き直してください。

cfg308 Warning : xxxx (line xxx): can't find <XXXX>
cfg308 Warning : xxxx (line xxx): can't find "XXXX"
インクルードファイル XXXX が見つかりません。ファイル名および存在を確認して下さい。

cfg308 Warning : over character number of including path-name
インクルードファイルのパス名が 255 文字を超えています。

6.3 makefile の編集

コンフィギュレータが生成した、makefile を編集し、コンパイルオプションやライブラリなどを設定します。以下にその設定方法を示します。

1. NC308WA コマンドオプション

   C コンパイラのコマンドオプションは"CFLAGS"に定義します。"-c"オプションは必ず指定して下さい。

2. AS308 コマンドオプション

   アセンブラのコマンドオプションは"ASFLAGS"に定義します。

3. LN308 コマンドオプション

   リンカのコマンドオプションは"LDFLAGS"に定義します。特に指定しなければならないオプションはありません。

4. ライブラリの指定

   ライブラリの指定は、"LIBS" に定義します。
   コンフィギュレータがコンフィギュレーションファイルとカレントディレクトリのソースファイルから必要なライブラリを"LIBS"に定義します。必要に応じて追加、削除を行って下さい。

独自に makefile を作成する場合は、下記 4 項目を必ず、メイクファイルに記述してください。

1. MR308 ライブラリの指定

   mr308.lib と c308mr.lib を指定する必要があります。

2. アセンブルオプションの指定

   サービスコールを発行しているアセンブリ言語で記述したソースファイルをアセンブルする場合には、必ず、 アセンブルオプション"-F"を指定してください。

3. リンクを行う前の処理

   リンクを行う前には、必ず、以下に示す 2 つの処理を以下に示す順序で実行してください。
   1. mr308tbl
   2. as308 mrtable.a30

   mr308tbl は、MR308 が用意しているユーティリティです。mr308tbl はコンフィギュレータ cfg308 を起動したディレクトリ（mr308tbl はカレントディレクトリの cfg308 が出力した mr308.inc、vector.tpl を読み込んで処理します）で起動してください。また、コンパイラ、アセンブラの生成するサービスコールファイル（XXX.mrc）が出力されるディレクトリ（r30 ファイルが出力されているディレクトリと同じ）がカレントディレクトリでない場合は、その出力先ディレクトリを mr308tbl の引数に指定して下さい。
   例) mr308tbl outputdir
   注）デバッガ PD308 でサービスコール発行機能を使用する場合は、mr308tbl 起動時の引数に$(LIB308)（MR308 インストール先ディレクトリの lib308）を追記してください。
   mr308tbl を実行すれば、mrtable.a30 ファイルがカレントディレクトリに生成されます。
   上記に示した処理を実行した後、mrtable.r30 ファイルを含めてリンクを実行してください。

6.4 make 実行時のエラー

make を実行時に mr308tbl が以下のワーニングを出力する場合があります。

mr308tbl Warning : You need not specify systime.timeout YES in configuration file

以下のサービスコールを使用しない場合は、コンフィギュレーションファイルのシステム定義で"Timeout=YES"を設定する必要がありません。

```
   tslp_tsk,twai_flg,trcv_mbx,twai_sem,tsnd_dtq,trcv_dtq,vtsnd_dtq,vtrcv_dtq,
tget_mpf
```

コンフィギュレーションのシステム定義で"Timeout=NO;"を設定すれば、このワーニングは消えます。上記 9 つのサービスコールを使用しない場合には"Timeout=NO;"を設定することで、RAM 使用量や、上記 9 サービスコール以外のカーネルコードサイズが若干減ります。

第 7 章　　アプリケーション作成の手引き

7.1 ハンドラからのサービスコールの処理手順

ハンドラ[61]からのサービスコール発行はタスクからのサービスコールと異なり、サービスコール発行時にタスク切り替えは発生しません。タスク切り替えが発生するのはハンドラからの復帰時です。

ハンドラからのサービスコール処理手順は大きく分けて以下の３通りがあります。

1. タスク実行中に割り込んだハンドラからのサービスコール
2. サービスコール処理中に割り込んだハンドラからのサービスコール
3. ハンドラ実行中に割り込んだ（多重割り込み）ハンドラからのサービスコール

[61] OS 独立割り込みハンドラからはサービスコールは発行できませんので、ここで述べているハンドラは OS 独立割り込みハンドラを含みません。

7.1.1 タスク実行中に割り込んだハンドラからのサービスコール

スケジューリング（タスク切り替え)は ret_int サービスコールによりおこなわれます。[62] (図 7.1参照)

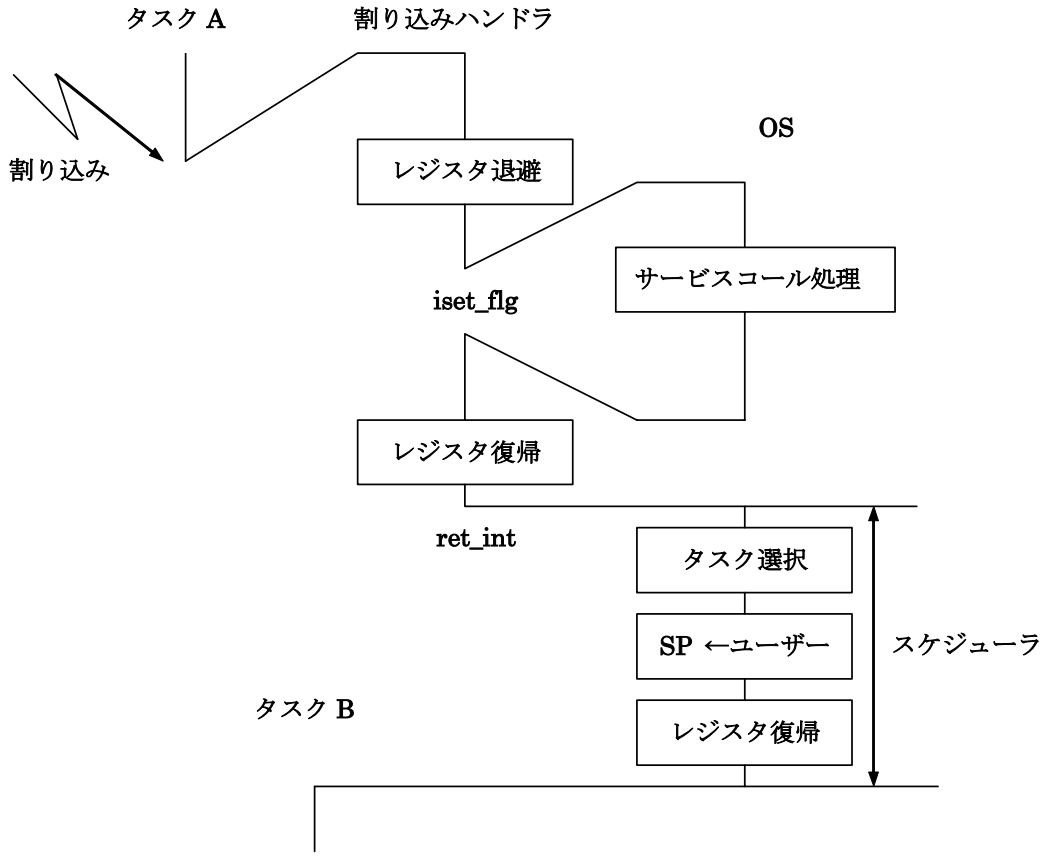


図 7.1 タスク実行中に割り込んだ割り込みハンドラからのサービスコール処理手順

[62] C 言語で OS 依存割り込みハンドラを記述する場合（#pragma INTHANDLER 指定時)、ret_int サービスコールは自動的に発行されます。

7.1.2 サービスコール処理中に割り込んだハンドラからのサービスコール

スケジューリング（タスク切り替え)は割り込まれたサービスコール処理に戻った後におこなわれます。（図7.2参照）

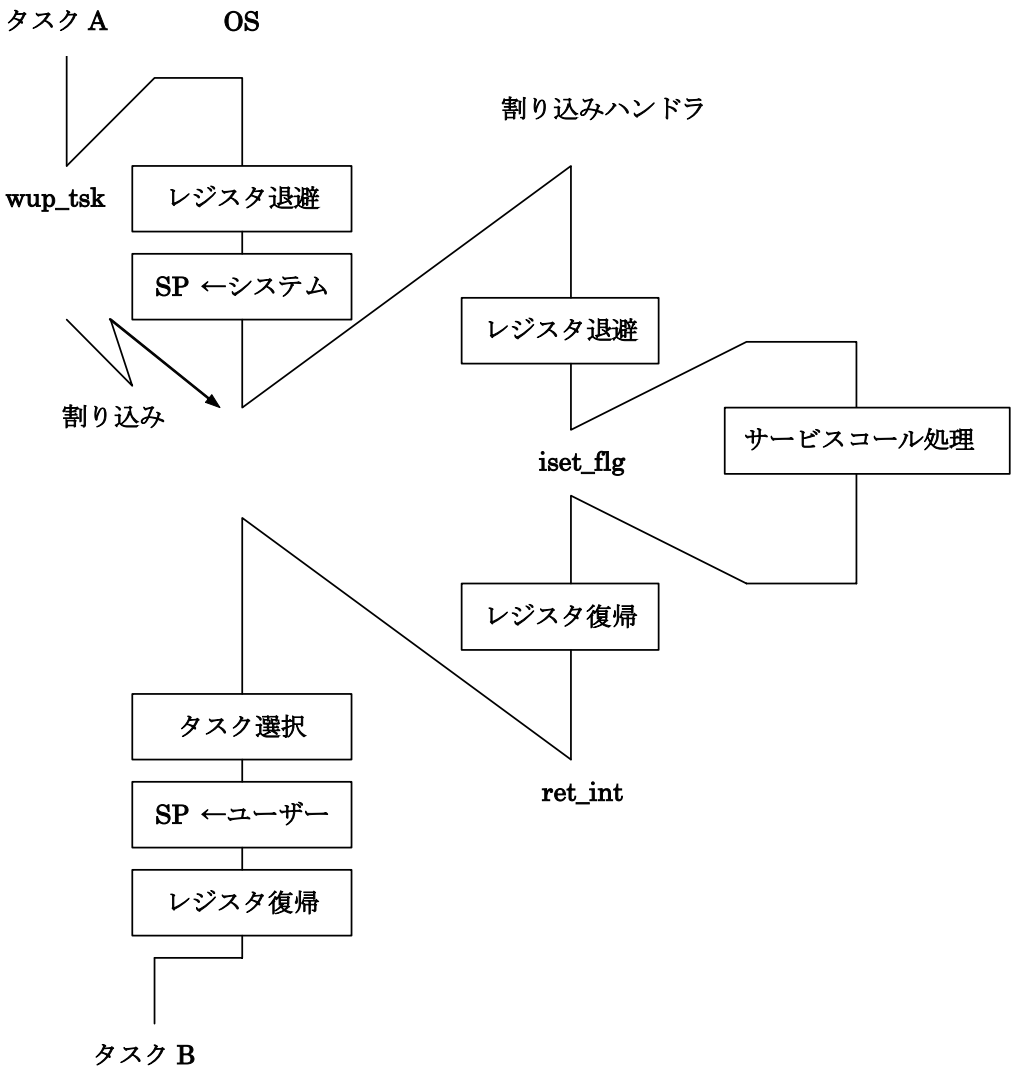


図 7.2　サービスコール処理中に割り込んだ割り込みハンドラからのサービスコール処理手順

7.1.3 ハンドラ実行中に割り込んだハンドラからのサービスコール

ハンドラ（以後ハンドラ A と呼びます。)実行中に割り込みが発生した場合を考えます。ハンドラ A 実行中に割り込んだハンドラ（以後ハンドラ B と呼びます。)が、発行したサービスコールによりタスク切り替えが必要になった場合は、ハンドラ B から復帰するサービスコール（ret_int サービスコール)では、ハンドラ A に戻るだけでタスク切り替えはおこないません。

ハンドラ A からの ret_int サービスコールによりタスク切り替えが行われます。（図 7.3参照)

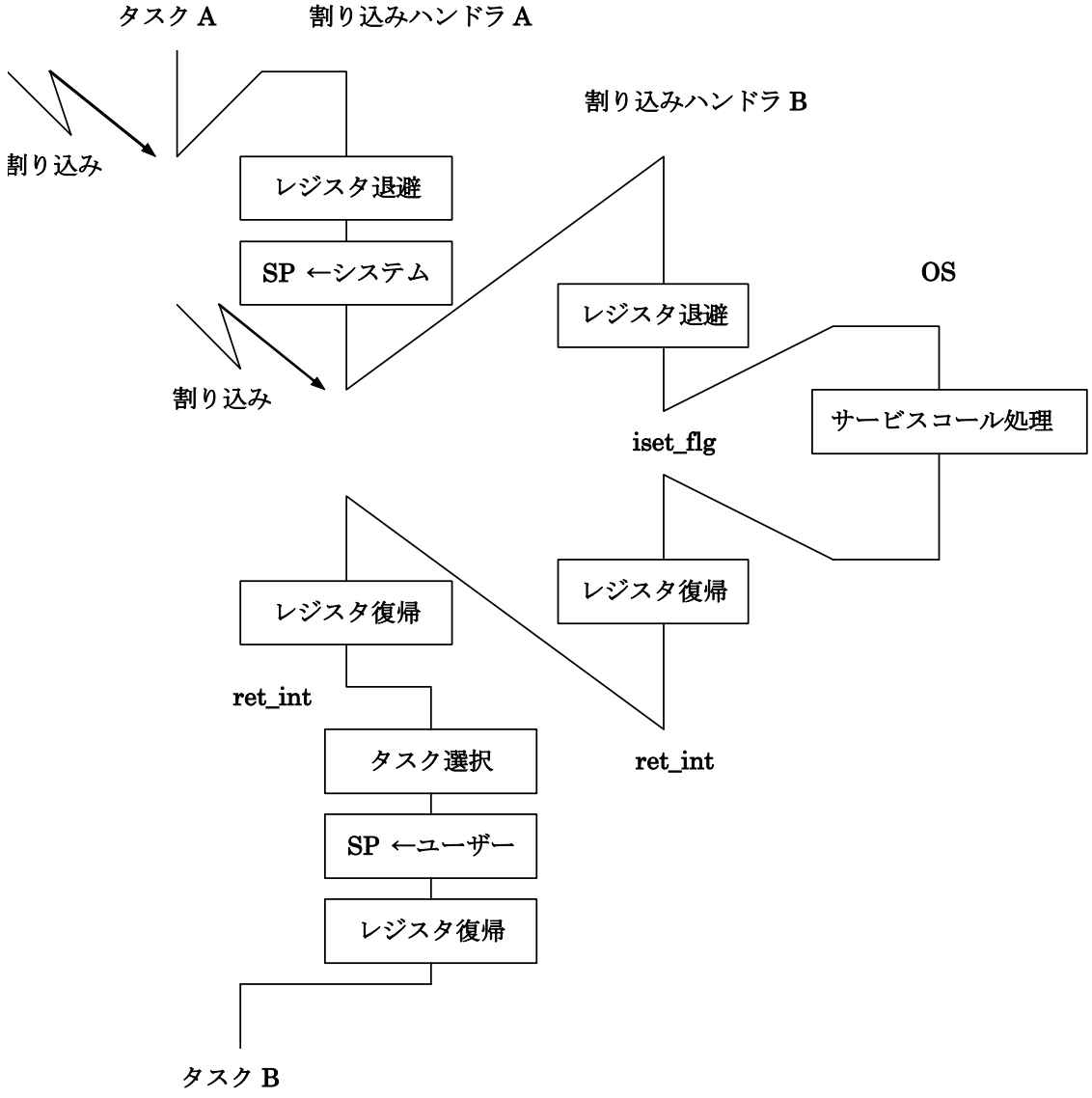


図 7.3 多重割り込みハンドラからのサービスコール処理手順

7.2 スタックについて

7.2.1 システムスタックとユーザースタック

MR308 のスタックにはシステムスタックとユーザースタックがあります。

- ユーザースタック
  タスクごとに 1 つずつ存在するスタックです。したがって MR308 を用いてアプリケーションを記述する場合はタスクごとのスタック領域を確保する必要があります。

- システムスタック
  MR308 内部（サービスコール処理中）に使用されるスタックです。MR308 ではサービスコールをタスクが発行するとスタックをユーザースタックからシステムスタックに切り替えます。(図 7.4を参照して下さい。)
  システムスタックは、割り込みスタックを使用します。

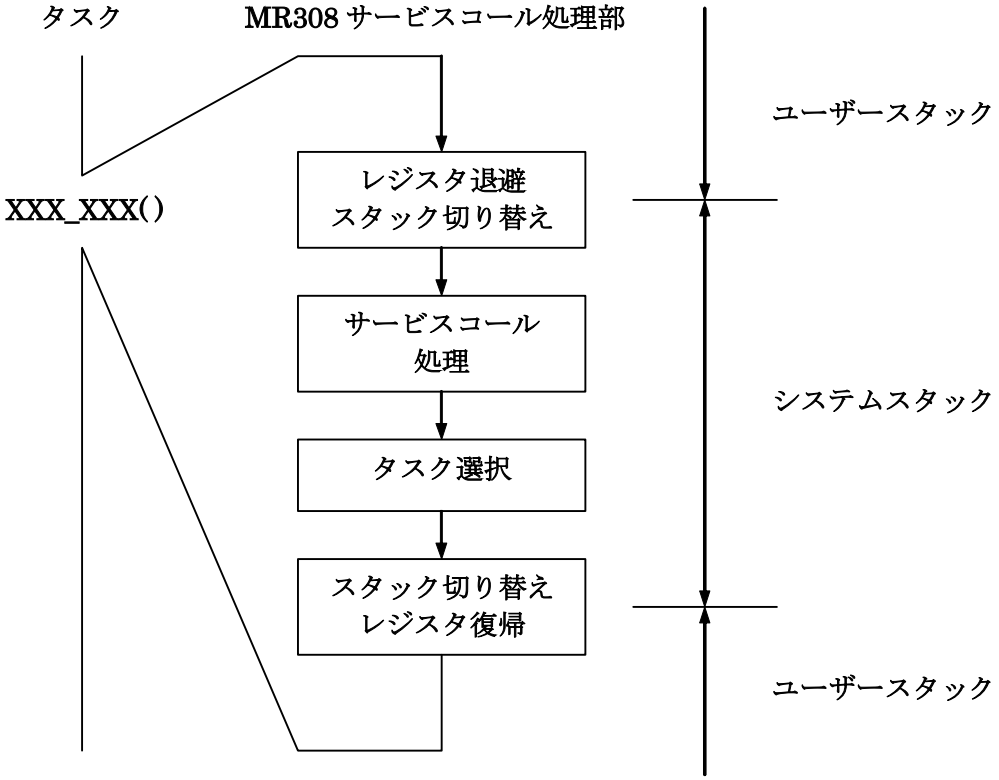


図 7.4　システムスタックとユーザースタック

また、ベクタ番号が 0～31、247～255 の割り込み発生時には、ユーザースタックからシステムスタックに切り替えます。したがって、割り込みハンドラで使用するスタックは全てシステムスタックを使用します。

第 8 章　　サンプルプログラムの説明

8.1 概要

MR308 の応用例として、タスク間で交互に標準出力に文字列を出力するプログラムを示します。

表 8-1　サンプルプログラムの関数一覧

関数名	種類	ID 番号	優先度	機能
main()	タスク	1	1	task1、task2 を起動させます。
task1()	タスク	2	2	“task1 running”を出力します。
task2()	タスク	3	3	“task2 running”を出力します。
cyh1()	ハンドラ	1		task1()を起床します。

以下に、処理内容を説明します。

- main タスクは、task1、task2、cyh1 を起動し、自タスクを終了させます。

- task1 は、次の順で動作します。
  1. セマフォを獲得します。
  2. 起床待ちに移行します。
  3. “task1 running”を出力します。
  4. セマフォを解放します。

- task2 は、次の順で動作します。
  1. セマフォを獲得します。
  2. “task2 running”を出力します。
  3. セマフォを解放します。

- cyh1 は、100ms 毎に起動し、task1 を起床します。

8.2 ソースプログラム

```
 1 /***********************************************************************************
 2 *                       MR308  smaple program
 3 *
 4 * COPYRIGHT(C) 2003(2005) RENESAS TECHNOLOGY CORPORATION
 5 * AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
 6 *
 7 *
 8 *     $Id: demo.c,v 1.2 2005/06/15 05:29:02 inui Exp $
 9 ***********************************************************************************/
10
11 #include <itron.h>
12 #include <kernel.h>
13 #include "kernel_id.h"
14 #include <stdio.h>
15
16
17 void main( VP_INT stacd )
18 {
19     sta_tsk(ID_task1,0);
20     sta_tsk(ID_task2,0);
21     sta_cyc(ID_cyh1);
22 }
23 void task1( VP_INT stacd )
24 {
25     while(1){
26             wai_sem(ID_sem1);
27             slp_tsk();
28             printf("task1 running¥n");
29             sig_sem(ID_sem1);
30     }
31 }
32
33 void task2( VP_INT stacd )
34 {
35     while(1){
36             wai_sem(ID_sem1);
37             printf("task2 running¥n");
38             sig_sem(ID_sem1);
39     }
40 }
41
42 void cyh1( VP_INT exinf )
43 {
44     iwup_tsk(ID_task1);
45 }
46
```

8.3 コンフィギュレーションファイル

```
 1 //************************************************************************
 2 //
 3 //   COPYRIGHT(C) 2003,2005 RENESAS TECHNOLOGY CORPORATION
 4 //   AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
 5 //
 6 //      MR308 System Configuration File.
 7 //        "$Id: smp.cfg,v 1.5 2005/06/15 05:41:54 inui Exp $"
 8 //
 9 //************************************************************************
10
11 // System Definition
12 system{
13        stack_size     = 1024;
14      priority        = 10;
15        system_IPL     = 4;
16      task_pause      = NO;
17      timeout         = YES;
18      tic_nume        = 1;
19      tic_deno        = 1;
20      message_pri     = 255;
21 };
22 //System Clock Definition
23 clock{
24      mpu_clock               = 20MHz;
25        timer                 = A0;
26        IPL                   = 4;
27 };
28 //Task Definition
29 //
30 task[]{
31      entry_address   = main();
32      name            = ID_main;
33        stack_size     = 100;
34      priority        = 1;
35      initial_start   = ON;
36 };
37 task[]{
38      entry_address   = task1();
39      name            = ID_task1;
40        stack_size     = 500;
41      priority        = 2;
42 };
43 task[]{
44      entry_address   = task2();
45      name            = ID_task2;
46      stack_size      = 500;
47      priority        = 3;
48 };
49
50 semaphore[]{
51      name            = ID_sem1;
52      max_count       = 1;
53      initial_count   = 1;
54      wait_queue      = TA_TPRI;
55 };
56
57
58
59 cyclic_hand [1] {
60      name                    = ID_cyh1;
61      interval_counter        = 100;
62      start                   = OFF;
63      phsatr                  = OFF;
64      phs_counter             = 0;
65      entry_address           = cyh1();
66      exinf                   = 1;
67 };
```

第 9 章　　別 ROM 化について

9.1 別 ROM 化の方法

本章では、MR308 のカーネルとアプリケーションプログラムを別の ROM に配置する方法を示します。
図 9.1は、2 つの異なるアプリケーション間のの共通部分とカーネル部分はカーネル ROM に、アプリケーション部分はそれぞれ別の ROM に配置した場合の例を示しています。
この例をもとに、ROM 分割の方法を示します。

1. システム構成

アプリケーションプログラムのシステム構成を行います。
ここでは、2 つのアプリケーションプログラムのシステム構成が以下に示すものとして説明を行います。

	アプリケーション 1	アプリケーション 2
タスク数	4	5
イベントフラグ数	1	3
セマフォ数	4	2
メールボックス数	3	5
固定長メモリプール数	3	1
周期ハンドラ数	3	3

2. コンフィギュレーションファイル作成

システム構成を行なった結果をもとに、コンフィギュレーションファイルを作成します。この時、maxdefine 定義部で設定する以下の項目を 2 つのコンフィギュレーションファイル間で共通にする必要があります。この maxdefine 定義部で設定する値は、2 つのアプリケーション間の各定義について定義数の大きい方を指定します。

例

```
maxdefine{
        max_task        = 5;
        max_flag        = 3;
        max_sem         = 4;
        max_mbx         = 5;
        max_mpl         = 3;
        max_cyh         = 3;
};
```

その他の定義については､2 つのコンフィギュレーションファイル間で異なっても問題ありません。

3. プロセッサモードレジスタの変更

スタートアッププログラムのプロセッサモードレジスタをシステムに応じて変更します。

4. アプリケーションプログラム作成

2 つのアプリケーションプログラムを作成します。

5. 各セクション配置設定

カーネル ROM とアプリケーション ROM に配置するプログラムを以下に示します。

- カーネル ROM に配置するプログラム
スタートアッププログラム（MR_KERNEL セクション）
MR308 のカーネル（MR_KERNEL セクション）
2 つのアプリケーション間の共通プログラム（program セクション）
固定割り込みベクタ領域（FIX_INTERRUPT_VECTOR セクション）

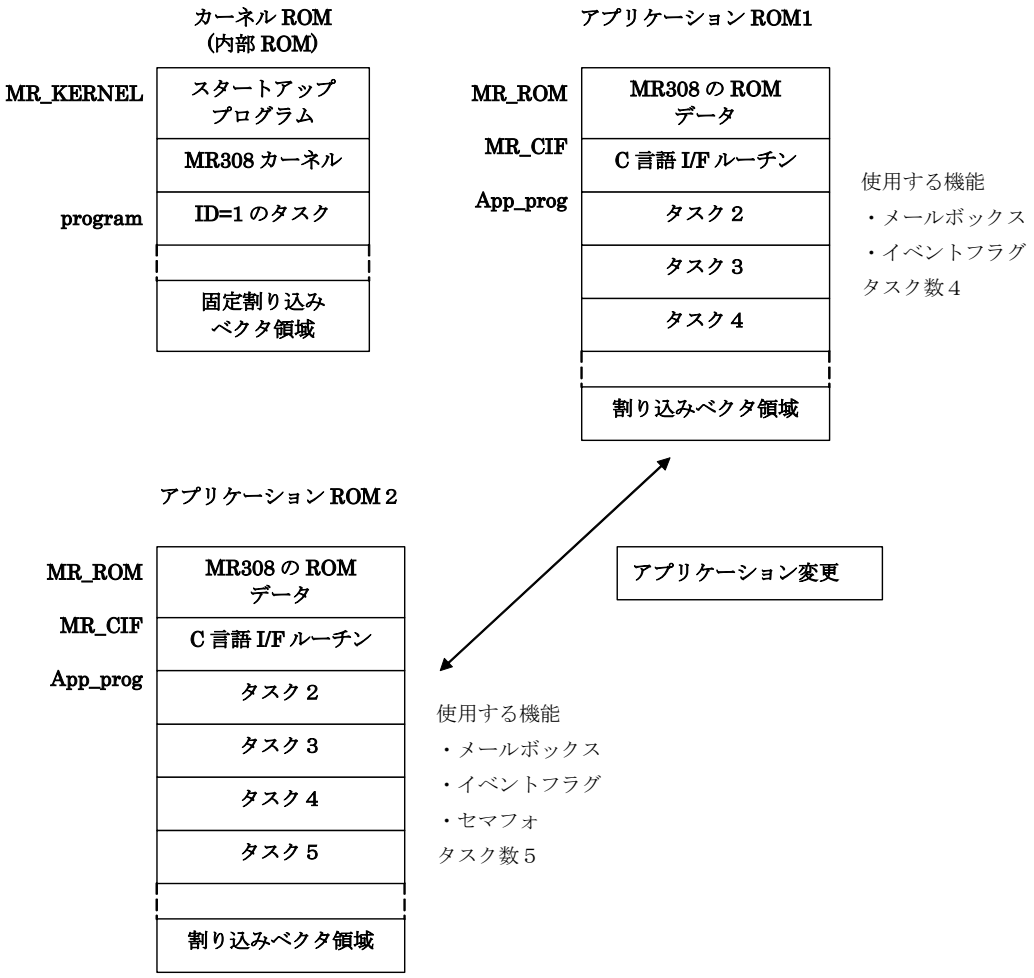


図 9.1　ROM 分割

- アプリケーション ROM に配置するプログラム
  MR308 の ROM データ（MR_ROM セクション）
  C 言語 I/F ルーチン（MR_CIF セクション）
  アプリケーションプログラム（app_prog セクション）
  割り込みベクタ領域（INTERRUPT_VECTOR セクション）

- 各プログラムの配置方法を以下に示す。
  ユーザープログラムのセクション名変更
  C 言語でアプリケーションプログラムを記述している場合は、以下に示すように #pragma SECTION を使ってアプリケーション ROM に配置するプログラムのセクション名を変更します。NC308WA では、ユーザープログラムのセクション名は、設定がなければ program セクションになります。そのため、アプリケーション ROM に配置するタスクを別のセクション名に変更しなければなりません。[63]

[63] カーネル ROM に配置するタスクのセクションは、変更する必要はありません。

```
#pragma SECTION program app_prog/* プログラムのセクションを変更 */
/* task2,task3 のセクション名は app_progに変わります。 */
void task2(void){
                :
}

void task3(void){
                :
}
```

セクションの配置設定

セクションファイル (c_sec.inc、asm_sec.inc)を変更して、アプリケーション ROM に配置するプログラムのアドレス設定を行ないます。この時、以下に示すセクションの開始アドレスは、2 つのアプリケーション間で一致していなければなりません。また、アプリケーション ROM の先頭には、必ず MR_ROM セクションを配置しなければなりません。その他のセクションの配置順序の決まりはありません。

- MR308 の ROM データ (MR_ROM セクション)
- C 言語 I/F ルーチン (MR_CIF セクション)
- 割り込みベクタ領域 (INTERRUPT_VECTOR セクション)

以下に、セクションファイルの設定例を示す。

```
.section MR_ROM,ROMDATA ; MR308のROMデータ
.org    0fe0000H                ; 2つのアプリケーション間で共通アドレス

.section MR_CIF,CODE    ; C 言語 I/F ルーチン
.section app_prog,CODE  ; ユーザープログラム

.section INTERRUPT_VECTOR       ; 割り込みベクタ
.org    0fefd00H                ; 2つのアプリケーション間で共通アドレス
```

図 9.2に示すメモリマップになります。

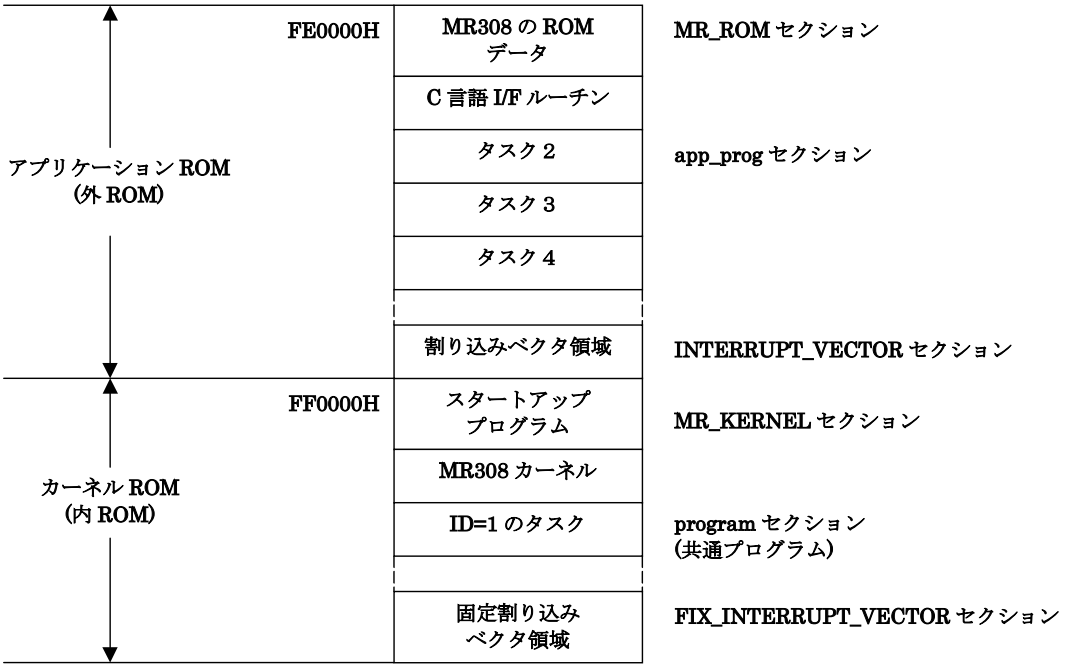


図 9.2　メモリマップ

6. コンフィギュレータ cfg308 を実行します。

7. makefile を編集します。

   makefile 中の mr308tbl に、引数を指定します。ここでの引数指定は、2 つのアプリケーションプログラムがあるディレクトリを指定します。
   アプリケーション 1 のディレクトリが /product/app1 で、アプリケーション 2 のディレクトリが /product/app2 の場合の例を以下に示します。

例

```
mr308tbl /product/app1 /product/app2
```

8. システム生成

   make コマンドを実行し、システム生成します。[64]

9. 4から8の処理をアプリケーション 2 についても行うことで、アプリケーション 2 のシステムが生成できます。

以上に述べた方法で、別 ROM 化が行なえます。

[64] カレントディレクトリに mrtable.a30 ファイルがない場合は、make を実行して生成します。

M16C/70,80,M32C/80 シリーズ用リアルタイム OS
ユーザーズマニュアル
M3T-MR308/4

発行年月日　2005 年 11 月 1 日　Rev.2.00

発行　　株式会社 ルネサス テクノロジ　営業企画統括部
〒100-0004　東京都千代田区大手町 2-6-2

編集　　株式会社 ルネサス ソリューションズ　第一応用技術部

M3T-MR308/4 V.4.00
ユーザーズマニュアル

ルネサス エレクトロニクス株式会社
神奈川県川崎市中原区下沼部1753 〒211-8668

RJJ10J0836-0200Z